Technische Universität Berlin
Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

Preserving Paths in Temporal Graphs

Bachelorarbeit

von Carsten Schubert

zur Erlangung des Grades „ Bachelor of Science “ (B. Sc.)
im Studiengang Informatik

Erstgutachter:	Prof. Dr. Rolf Niedermeier
Zweitgutachter:	Prof. Dr. Martin Skutella
Betreuer:	Till Fluschnik, Prof. Dr. Rolf Niedermeier, Philipp Zschoche

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.
The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, 30th September

.....
Signature

Zusammenfassung

In dieser Arbeit stellen wir das sogenannte Multistage st-Pfad-Problem vor und untersuchen seine parameterisierte Komplexität. Gegeben ist dabei ein temporaler Graph $G = (V, E_1, \ldots, E_\tau)$, zwei spezielle Knoten $s, t \in V$ und zwei natürliche Zahlen k und ℓ. Es wird dann die Frage gestellt, ob in jeder "Schicht" $G_i = (V, E_i)$ von G ein Pfad P_i von s nach t mit Länge höchstens ℓ existiert, sodass alle aufeinanderfolgenden Pfade P_i, P_{i+1} jeweils zueinander eine Distanz von höchstens k haben. Wir untersuchen drei verschiedene solche Distanzen und folglich auch drei Varianten des Problems: Die Größe der symmetrischen Differenz der Pfadknoten ("Knotendistanz"), die Größe der symmetrischen Differenz der Pfadkanten ("Kantendistanz"), und die Levenshtein-Distanz.

Wir zeigen, dass alle drei Problemvarianten NP-schwer sind, sogar für beschränkte Instanzen (z.B. wenn k konstant ist). Außerdem beweisen wir, dass die Variante mit der Knotendistanz nicht in subexponentieller Zeit (bezogen auf die Anzahl gegebener Knoten) entschieden werden kann, es sei denn die Exponentialzeithypothese wird widerlegt.

Wir zeigen weiterhin, dass alle drei Varianten bezüglich des Parameters $k + \ell + \tau$ W[1]-schwer sind. Die Varianten mit Knoten- und Kantendistanz sind ebenfalls W[1]-schwer bezüglich des Parameters $\nu + k + \ell$, wobei ν die Knotenüberdeckungszahl des unterliegenden Graphen $G_\downarrow$ von G darstellt.

Demgegenüber konnten wir aber auch einige positive Ergebnisse feststellen: So geben wir beispielsweise einen Problemkern für alle drei Problemvarianten an, dessen Größe nur von dem zusammengesetzten Parameter $\nu + \tau$ abhängt. Für die dazu gehörigen Datenreduktionen führen wir das Konzept "temporaler Zwillinge" ein. Leider haben wir auch festgestellt, dass ein Problemkern polynomieller Größe bezüglich $\nu + \tau + k + \ell$ für die Variante mit Knotendistanz nicht existieren kann, es sei denn coNP $\subseteq$ NP/poly. Ferner geben wir einen FPT-Algorithmus für alle drei Problemvarianten für den Parameter $\Delta + \ell$ an, wobei Δ den Maximalgrad über alle Schichten von G darstellt. Als letztes zeigen wir einen Problemkern von Multistage st-Pfad mit Knotendistanz, dessen Größe polynomiell im Parameter $\rho + \tau$ ist. Hierin ist ρ die Größe eines minimalen "Feedback Edge Sets" des unterliegenden Graphen von G.

Abstract

In this work, we introduce and study the computational and parameterised complexity of the so-called Multistage st-Path problem: Given a temporal graph $G = (V, E_1, \ldots, E_\tau)$, two special vertices $s, t \in V$ and two integers $k, \ell \in \mathbb{N}$, the question is whether there is an st-path P_i of length at most ℓ in each temporal graph layer $G_i = (V, E_i)$ such that all consecutive paths P_i, P_{i+1} have a distance of at most k towards each other. We investigate three different types of such distances, namely the size of the symmetric difference of path vertices ("vertex distance"), the size of the symmetric difference of path edges ("edge distance"), and the Levenshtein distance.

On the negative side, we show that all three problem variants are NP-hard even on restricted inputs (e.g. if k is constant). We also show that Multistage st-Path with vertex distance cannot be solved in time subexponential in its number of vertices, unless the Exponential Time Hypothesis fails.

We show that all three problem variants are W[1]-hard when parameterised by $\tau + k + \ell$. Moreover, the variants with vertex and edge distance we prove to be W[1]-hard when parameterised by $\nu + k + \ell$, where ν is the vertex cover number of the underlying graph $G_\downarrow$ of G.

On the positive side, we provide a problem kernelisation for all three problem variants with respect to the combined parameter $\nu + \tau$. To this end we introduce the concept of temporal twin vertices, which allow for efficient preprocessing in our case. Unfortunately, a problem kernel of size polynomial in $\nu + \tau + k + \ell$ for Multistage st-Path with vertex distance does not exist, unless coNP $\subseteq$ NP/poly. We then provide an FPT-algorithm for all three problem variants with respect to the parameters Δ and ℓ, where Δ is the maximal degree over all input layers. At last, we show a problem kernel for Multistage st-Path with vertex distance, which is of size polynomial in $\rho + \tau$, where ρ is the feedback edge number of $G_\downarrow$.

Contents

1 Introduction

Imagine we are in charge of managing an expanding international logistics company, mainly specialised in transporting large valuable goods around the globe for corporate clients with long-term contracts. As such, we often organise cargo transportation between a limited number of locations. We have to coordinate and supervise these shipments along different means of carriage (e.g. trains, convoys, and container ships) on different track sections, since there is no universal transport method for the entire way that is appropriate for our goods. Then in order to ensure safe transportation we also need to manage offices at intermediary points between freight origin and destination or hire sub-contractors to handle this for us. Of course, we are interested in finding a way which involves preferably few of such maintained offices or sub-contractors in order to reduce costs. This usually also aligns with our goal of delivering cargo fast and thus satisfying our customers.

We may encounter the problem that we are tasked with a delivery at some point in time where some track sections are not available for our purposes. For instance, some means of carriage might only run at scheduled times, like trains or ships. Roads might be periodically blocked due to high traffic density or unfavourable weather, not to mention the likeliness of accidents. Sub-contractors may also be incapable of handling our requests at certain times—at least at rates reasonable to us—due to their work policies or simply good order situations (think of e.g. christmas business). In short, if we still want to deliver our goods fast and reliably in such situations, then we need to be flexible for potentially re-routing the used paths at different points in time. But that may require additional offices or partnerships to sub-contractors, naturally leading to increased expenses, which we still try to keep as low as possible.

In this work, we model such a situation with a time-dependent-network, i.e. a network whose links may be active or inactive at different limited fixed points in time. The nodes in this network then refer to fixed locations (that are e.g. suited for an intermediary office) and the time-dependent links refer to track sections between these locations which are available to us at certain points in time. An example of such a model for our situation with two points in time is shown in Figure 1.1.

The Multistage st-Path problem then informally asks, given such a time-dependent network and two nodes in this network, whether there are short paths between the given nodes at all times such that these paths are somewhat "similar" from each point in time to the consecutive one. In the application above, the required path similarity results from the intention of minimising not only the number of nodes in each time-dependent path, but also the number of changed nodes between alternative routes taken at different times. We will research this problem regarding its computational and parameterised complexity by showing (parameterised) hardness results as well as providing FPT-algorithms and kernelisations.

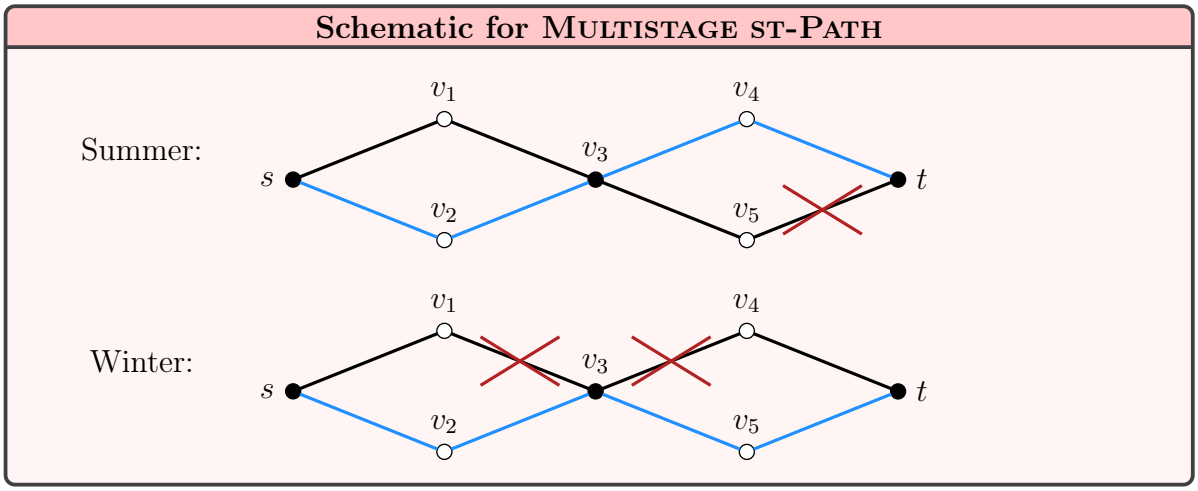


Figure 1.1: Illustration of an example transportation network as described in the introduction. We plan to deliver cargo from location s to t at different points in time, some in summer and some in winter. Unfortunately in both seasons some track sections are unavailable to us, as indicated by the crossing signs. This especially means that we are not capable of routing our delivery via location v_5 in summer, or via locations v_1 and v_4 in winter (at least if we do not want to take pointless detours). Since we still want our tracks to be as similar as possible in both seasons, we decide to transport on the ways that are indicated by blue lines: We use the tracks from s via v_2 to v_3 in summer and winter. Subsequently, we route our cargo via v_4 in summer and via v_5 in winter, as we have no other choice in order to reach t at both times.

1.1 Problem Formulation

We now formally describe Multistage st-Path, the main decision problem of this work:

Definition 1.1: Multistage st-Path

Input: A temporal graph $G = (V, E_1, \dots, E_\tau)$, two special vertices $s, t \in V$, and two integers $k, \ell \in \mathbb{N}$.

Question: Are there paths $(P_1, \dots, P_\tau)$ so that P_i is an st-path in $G_i = (V, E_i)$ for all $i \in [\tau]$, $|V(P_i)| \leq \ell$ for all $i \in [\tau]$, and $\text{dist}(P_i, P_{i+1}) \leq k$ for all $i \in [\tau - 1]$?

Herein, dist can be any function that somehow measures a distance between two paths. In this work, we will study the following distance functions (and thus variants of Multistage st-Path):

(1) Vertex distance, i.e. $\text{dist}(P_x, P_y) = |V(P_x) \triangle V(P_y)|$

(2) Edge distance, i.e. $\text{dist}(P_x, P_y) = |E(P_x) \triangle E(P_y)|$

(3) Levenshtein distance, i.e. $\text{dist}(P_x, P_y)$ is the minimum number of edits between P_x and P_y regarded as vertex strings, where each edit is either a single vertex insertion, a single vertex deletion or a single vertex substitution (Yujian and Bo 2007).

The first problem variant thus considers the symmetric difference of the vertex sets of two consecutive paths, the second variant addresses the symmetric difference of the edge sets of two consecutive paths, and the last variant compares the number of vertex edits between consecutive paths. From now on, we will often abbreviate the names of those problem variants: Vertex-MstP stands for Multistage st-Path if it uses the vertex distance, Edge-MstP is short for Multistage st-Path which uses the edge distance, and Leven-MstP abbreviates Multistage st-Path when the Levenshtein distance is used.

If P_x and P_y are two paths of length at most ℓ, their vertex or edge distance can trivially be computed in $O(\ell^2)$ time. There is also an appropriate algorithm which computes the Levenshtein distance of P_x and P_y in $O(\frac{\ell^2}{\log \ell})$ time (Masek and Paterson 1980). Thus, all listed distances can be computed in $O(\ell^2)$ time on paths of length at most ℓ.

We define the tuple of paths $S = (P_1, \dots, P_\tau)$ as a solution for an instance of Multistage st-Path if the conditions listed above are met by those paths. Two paths P_x and P_y in S are called adjacent if $|x - y| = 1$.

It is easy to see that Vertex-MstP, Edge-MstP, and Leven-MstP are all in NP, since each solution is of size polynomially in the corresponding input size and verifying a solution S can be done in $O(\tau \ell^2)$ time. In order to do this, one first goes through each path P_x in S and checks whether P_x is indeed an st-path of length at most ℓ in the corresponding temporal graph layer G_x. Afterwards, one computes the distance between each two adjacent paths in S and checks whether it is at most k each time.

1.2 Related Work

This work follows the recent trend initiated by Gupta, Talwar, and Wieder (2014) and Eisenstat, Mathieu, and Schabanel (2014) of studying time-dependent, so-called "multistage" generalisations of already studied decision problems. In these multistage settings, one is not only given one but τ many instances—one for each point in time—of the underlying problem. The goal is then still to find a solution for each given instance of the underlying problem, but also to reach a certain "stability" (or similarity) between solutions corresponding to consecutive points in time.

Particularly, Eisenstat, Mathieu, and Schabanel (2014) considered a multistage generalisation of facility location problems, with their main result being a logarithmic approximation algorithm (later improved to a constant factor approximation by An, Norouzi-Fard, and Svensson (2017)). Gupta, Talwar, and Wieder (2014) described Multistage Maintenance Matroid problems, both in online and offline settings. Among other results, they formulated a logarithmic approximation algorithm for their offline version (which is optimal unless $\mathsf{P} = \mathsf{NP}$) and proved the Perfect Matching Maintenance problem to be NP-hard to approximate. They left open whether a special case, Bipartite Matching Maintenance, was hard to approximate as well, which was affirmed later by Bampis et al. (2018), among other approximation results. Bampis et al. (2019) recently also introduced a framework for online multistage maximisation problems, but these results do not directly relate to our setting, mainly because we exclusively consider an offline problem. More important to us, Gupta, Talwar, and Wieder (2014) already suggested in the concluding section of their aforementioned work to study polynomial-time solvable problems such as shortest path in a multistage setting, which is precisely what we are going to do in this work.

In contrast to these previously listed works, we focus on exact algorithms, employing methods from parameterised algorithmics. As such, our research is also closely related to the study of Multistage Vertex Cover by Fluschnik et al. (2019), which was mainly focused on parameterised complexity with respect to input parameters, such as solution size of the classical Vertex Cover problem or the number of layers in the input instance. A difference between their and our setting, however, is that the "single-stage" case of their problem, Vertex Cover, is already NP-hard, whereas Shortest st-Path (the single-stage version of our problem) is polynomial-time solvable.

In our case, we will thus show that generalising Shortest st-Path by a multistage setting yields an NP-hard problem. This does not seem surprising, since many variants of Shortest st-Path with additional requirements have been proven to be NP-hard as well (see e.g. Hassin (1992), Yu and Yang (1998), and Van Bevern, Fluschnik, and Tsidulko (2018)). Many variations of Shortest st-Path problems have also been studied extensively on dynamic and temporal graphs. These studies involve e.g. finding short weighted paths with dynamically changing edge weights (Ahuja et al. 2003), applications for time-dependent vehicle routing (Malandraki and Daskin 1992), and finding several variations of short paths emerging over time in temporal graphs as described by Wu et al. (2014). A path in their context however includes multiple edges corresponding to different points in time, since it "takes time" to traverse each edge along a path.

Table 1.1: Overview on our results. Each cell corresponds to a statement about one of our three studied problem variants with respect to a certain parameter. The vertex cover number of the underlying graph of the input temporal graph is referred to as ν. The feedback edge number of the underlying graph of the input temporal graph is referred to as ρ. The maximal degree over all input layers is denoted by Δ and tw denotes the maximal treewidth of all individual input layers. The other researched parameters are specified in Definition 1.1. NoPK abbreviates that no problem kernel of size polynomial in the respective parameter can be found unless coNP $\subseteq$ NP/poly. On the other hand, PK denotes that there is a problem kernel of size polynomial in the corresponding parameter.

	Vertex-MstP	Edge-MstP	Leven-MstP
k	NP-hard even if $k = 0$ (Cor. 2.20)	NP-hard even if $k = 4$ (Cor. 3.9)	NP-hard even if $k = 2$ (Cor. 3.9)
ℓ	W[1]-hard (Thm. 3.1)	W[1]-hard (Cor. 3.8)	W[1]-hard (Cor. 3.8)
τ	NP-hard even if $\tau = 2$ (Thm. 2.11)	NP-hard even if $\tau = 2$ (Thm. 2.1)	W[1]-hard (Cor. 3.8)
$k + \ell + \tau$	W[1]-hard (Thm. 3.1)	W[1]-hard (Cor. 3.8)	W[1]-hard (Cor. 3.8)
$k + \ell + \nu$	W[1]-hard (Thm. 3.10)	W[1]-hard (Thm. 3.16)	
$\tau + \nu$	FPT (Thm. 4.1), NoPK (Thm. 4.8)	FPT (Cor. 4.7)	FPT (Cor. 4.7)
Δ	NP-hard even if $\Delta = 3$ (Thm. 2.11)	NP-hard even if $\Delta = 3$ (Thm. 2.1)	
$\Delta + \ell$	FPT (Thm. 4.18)	FPT (Thm. 4.18)	FPT (Thm. 4.18)
$\rho + \tau$	PK (Thm. 4.21)		
tw	NP-hard even if $\tau = 2$ (Thm. 2.11)	NP-hard even if $\tau = 2$ (Thm. 2.1)	

To the best of our knowledge, we are the first who consider "preserving" paths over time, that is, to find one short paths at each fixed point in time, but additionally require such paths of consecutive times to be "similar" to each other to some extent.

1.3 Our Contributions

Table 1.1 summarises our parameterisation results for the three studied variants of Multistage st-Path. We especially highlight that Vertex-MstP and Edge-MstP are both W[1]-hard with respect to the parameters τ and ν (as can be derived from the W[1]-hardness for $k + \ell + \nu$). However, when parameterised by the combined parameter $\tau + \nu$, these problems are both fixed-parameter tractable.

1.4 Preliminaries

We denote by $\mathbb{N}$ and $\mathbb{N}^+$ the natural numbers including and excluding zero, respectively. The set of integers is denoted by $\mathbb{Z}$ and the set of rational numbers is denoted by $\mathbb{Q}$. We denote a set containing all natural numbers from 1 to n by $[n]$, i.e. $[n] = \{x \in \mathbb{N}^+ : x \leq n\}$ for any $n \in \mathbb{N}$. Similarly a set of all natural numbers from a to b is denoted by $[a, b]$, i.e. $[a, b] = \{x \in \mathbb{N} : a \leq x \leq b\}$. The symmetric difference of two sets A and B is denoted by $A \triangle B = (A \setminus B) \cup (B \setminus A)$. If x is a number, then we denote the smallest integer y so that $x \leq y$ by $\lceil x \rceil$. If x and y are both vectors with n elements, then we denote the standard scalar product of them by $x \cdot y$, i.e. $x \cdot y = \sum_{i=1}^{n} x_i y_i$. We also denote the maximum metric of x by $\|x\|_\infty$, i.e. $\|x\|_\infty = \max_{i=1}^{n} |x_i|$, and the sum metric of x by $\|x\|_1$, i.e. $\|x\|_1 = \sum_{i=1}^{n} |x_i|$. By log we denote the logarithm of base two.

1.4.1 Parameterised Complexity

Let Σ be a finite alphabet. In classical complexity theory, a *problem* is defined as a language $L \subseteq \Sigma^*$. Given an *instance* $x \in \Sigma^*$ of L the task then arises to decide whether x is a YES-instance, i.e. $x \in L$, or a NO-instance ($x \notin L$). Two instances x, x' of problems L, L' are *equivalent* if $x \in L \Leftrightarrow x' \in L$.

In parameterised complexity theory, we study problems with respect to some problem *parameter* $k \in \mathbb{N}$. A *parameterised problem* L is then a subset $L \subseteq \{(x, k) \in \Sigma^* \times \mathbb{N}\}$. Again, an instance (x, k) of L is a YES-instance if $(x, k) \in L$ and a NO-instance otherwise. We also call two instances $(x, k), (x', k')$ of parameterised problems L, L' equivalent if $(x, k) \in L \Leftrightarrow (x', k') \in L$. We call a parameterised problem L fixed-parameter tractable (FPT) if there is an algorithm deciding for each input instance (x, k) if it is a YES-instance of L in $f(k) \cdot |x|^{O(1)}$ time, where f is some computable function which only depends on k. A W[1]-hard problem is not fixed-parameter tractable unless W[1] = FPT.

In classical complexity theory we often show NP-hardness of a problem L' by describing a polynomial-time many-one reduction from an NP-hard problem L to L', i.e. an algorithm that takes an input instance x of L and then generates in $O(|x|^{O(1)})$ time an instance x' of L' such that x and x' are equivalent. This equivalence property is called *correctness* of the reduction.

Similarly, in parameterised complexity we often show W[1]-hardness of a parameterised problem L' by describing a *parameterised reduction* from a W[1]-hard problem L to L'. This is an algorithm which takes an input instance (x, k) of L and computes in $f(k)|x|^{O(1)}$ time an equivalent instance (x', k') of L' for which it additionally holds that k' is upper-bounded by $g(k)$, where both f and g are computable functions only depending on k. A *polynomial parameter transformation* is a special type of parameterised reduction which, after given its input instance (x, k), only takes $O((|x| + k)^{O(1)})$ time to output (x', k') and k' is polynomially upper-bounded in k.

A *kernelisation* of a parameterised problem L is an algorithm that, when given an input instance (x, k) of L, generates an equivalent instance (x', k') of L in time polynomial in $|x| + k$ such that $|x'| + k' \leq f(k)$, where f is some computable function that only depends on k. We call (x', k') a problem kernel with size f. Often such a kernelisation consists of

a set of preprocessing rules which we will call *data reductions*. A data reduction is *safe* if its input and output instances are equivalent.

1.4.2 Classic Graph Theory

A *simple* graph $G = (V, E)$ is a structure which contains a finite non-empty set V of *vertices* and a finite set $E \subseteq \{\{u, w\} : u \in V, w \in V, u \neq w\}$ of *edges*. We denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. We also say that G *includes* a vertex v or an edge e if $v \in V(G)$ or $e \in E(G)$, respectively. Vertices u and w are called *endpoints* of the edge $\{u, w\}$. A simple graph $G' = (V', E')$ is a subgraph of another simple graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. We then also say that G' is (contained) *in* G. A subgraph is *complete* or a *clique* if its edge set is of maximal size.

We will denote the set of *neighbours* of a vertex v in a simple graph G by $N(v) = \{w \in V : \{v, w\} \in E\}$. The *closed neighbourhood* of v, defined as the neighbours of v together with v itself will be denoted by $N[v]$, i.e. $N[v] = N(v) \cup \{v\}$. If it is unclear which graph we are referring to, we will specify this graph explicitly, like $N_G(v)$ or $N_G[v]$. The degree of v is denoted by $\deg(v)$ and is defined as the number of neighbours of v, i.e. $\deg(v) = |N(v)|$. Again, $\deg_G(v)$ may also be used. If $\deg(v) = 0$ for some vertex v in a simple graph, we call v an *isolated vertex*. The maximal degree Δ of a simple graph G is defined as $\max_{v \in V(G)} \deg(v)$.

A *path* P is a simple graph for which it holds that all n included vertices can be listed as a sequence in the form $v_1, v_2, \ldots, v_n$ so that $E(P) = \{\{v_i, v_{i+1}\} : i \in [n-1]\}$. We call P an *st*-path or a path *between* s and t if $s = v_1$ and $t = v_n$. The length of a path P is the number of its vertices, i.e. $|V(P)|$.

A simple graph G is *connected* if there is a path in G between each two distinct vertices $u, w \in V(G)$. A (connected) *component* C of G is an inclusion-maximal connected subgraph of G. Note that if G is connected, it contains only one component. An *st*-separator $S \subseteq V(G)$ in G for two distinct vertices $s, t \in V(G)$ is a set of vertices such that after removing all vertices in S (and their edges) from G there is no *st*-path in G left.

A *bipartition* $B = (L, R)$ of a simple graph G is a tuple of two vertex sets $L, R \subseteq V(G)$ so that $L \cup R = V(G)$, $R \cap L = \emptyset$, and for each edge $e \in E(G)$ there is one endpoint of e in L and the other one in R. A simple graph which has a bipartition is called a *bipartite graph*.

A simple graph is called *outerplanar* if it has an embedding in the plane where no edges cross each other and all vertices are on the same face, that is, no vertex is fully enclosed by other vertices and edges.

A simple graph is called *series-parallel* if it can be constructed by repeated application of *series expansions* and *parallel expansions* on its edges, starting from any number of paths with at most two vertices each. A series expansion is defined as a subdivision of any edge $\{a, b\}$ by a new vertex, that is, introducing a new vertex c and replacing $\{a, b\}$ by $\{a, c\}$ and $\{c, b\}$. A parallel expansion of any edge $\{a, b\}$ is defined as introducing a new vertex c adjacent to both a and b, i.e. adding the edges $\{a, c\}$ and $\{c, b\}$. All outerplanar graphs are series-parallel as shown by Duffin (1965). Moreover, it is well-known that

a simple graph is series-parallel if and only if is is of *treewidth* at most two (see e.g. Bodlaender and Antwerpen - de Fluiter (2001)).

A *vertex cover* of a simple graph G is a subset $C \subseteq V(G)$ such that each edge included in G has an endpoint in C. The *vertex cover number* of a simple graph G is the minimal number ν so that there is a vertex cover of G of size ν. A *feedback edge set* of a simple graph G is a subset $F \subseteq E(G)$ such that if all edges in F are removed from G the remaining simple graph is a forest, i.e. a simple graph with no cycles. The minimum size of which there is a feedback edge set in G is called feedback edge number of G.

From now on, we will refer to simple graphs often only as *graphs*.

1.4.3 Temporal Graph Theory

We also need the notion of *temporal graphs*, as our studied problems are defined on such structures. For our purposes, a temporal graph $G = (V, E_1, \ldots, E_\tau)$ consists of a finite non-empty vertex set V and τ many edge sets $E_1, \ldots, E_\tau \subseteq \{\{a, b\} : a \in V, b \in V, a \neq b\}$. Intuitively, in comparison to a simple graph with only one vertex and one edge set, a temporal graph contains τ many simple graphs which are ordered and all share the same vertex set.

We will refer to the graph $G_i = (V, E_i)$ as the i-th *layer* of the temporal graph G from now on. Layers G_i where i is an even number will occasionally be referred to as *even layers*, whereas layers G_i where i is an odd number are *odd layers*, respectively. We describe two layers G_i, G_j of a temporal graph as being *adjacent*, if $|i - j| = 1$. An edge included in some layer of a temporal graph is often referred to as *temporal edge*. Moreover, we will call $G_\downarrow = (V, \bigcup_{i=1}^{\tau} E_i)$ the *underlying graph* of a temporal graph G.

2 NP-Hardness

All described problem variants of Multistage st-Path are within the complexity class NP. In this chapter, we prove that Edge-MstP and Vertex-MstP are also NP-hard (and consequently, NP-complete). We do this by first showing the NP-hardness of Edge-MstP in Section 2.1. In the subsequent Section 2.2, we describe a polynomial-time reduction from Edge-MstP to Vertex-MstP in order to then deduce the NP-hardness of Vertex-MstP. We then strengthen this result by giving a larger running time lower bound for solving Vertex-MstP by assuming the Exponential Time Hypothesis in Section 2.3. NP-hardness of Leven-MstP remains unproven in this section, but will be shown later in Corollary 3.9 of Section 3.1.

2.1 NP-Hardness of Edge-MstP

In this section, we prove the following:

Theorem 2.1: NP-hardness of Edge-MstP

Edge-MstP is NP-hard, even if $\tau = 2$, both input layers are outerplanar graphs and the maximal degree in $G_\downarrow$ is 3.

In order to do so, we will reduce from the following problem which is one of the 21 problems proven to be NP-complete by Karp (1972):

Definition 2.2: Vertex Cover (VC)

Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a subset C of V so that $|C| \leq k$ and for each edge $\{a, b\} \in E$ there is $a \in C$ or $b \in C$?

The set E is assumed to be non-empty. From an instance of VC we derive an instance of Edge-MstP as described in the following Construction 2.3.

In this construction, a temporal graph G' with two layers is built. Per edge $e_j = \{a, b\}$ in the input graph, G' contains two "chains" of vertices, one for $a \in e_j$ and one for $b \in e_j$. These chains are then connected differently in the two layers, but any solution for Edge-MstP of the output instance needs to include the same chains in both paths. In the first layer G_1, we ensure that exactly one of the two chains for e_j is in each st-path in G_i. With the second layer we then ensure that the number of vertices in the input graph corresponding to such chains included to an st-path in G_2 does not exceed k, i.e. that a

solution for Edge-MstP of the output instance corresponds to a vertex cover of size at most k in the input instance.

Construction 2.3. Let $((V, E), k)$ be an instance of VC, where $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$. Let $k' = (2k + 3)m + n + 4$. Let $\ell = 2mn + m(k' + 2) + n + 42$. Parameters ℓ and k' will be used in the output instance. Next we construct some vertex sets.

Let $U_j = \{j_h^i : i \in e_j, h \in [k' + 2]\}$ for each $j \in [m]$. Let $U = \bigcup_{j \in [m]} U_j$.

These vertices will later form the aforementioned "chains", i.e. two separate paths for each edge in E. Let

$$X = \{X_j : j \in [m + 1]\},\ Y = \{Y_j^i : i \in [n], j \in [m]\} \text{ and } Z = \{Z_j^i : i \in [n], j \in [m]\}.$$

We will use the vertices in X to "route" all st-paths in the first layer of G', whereas vertices Y and Z will be used in the second layer as "connection points" for the chains formed by vertices in U. Let $V' = V \cup X \cup Y \cup Z \cup U \cup \{s, t, v_{n+1}\}$. This set is then used as vertex set of our constructed temporal graph G'.

As already indicated, G' shall have two layers. To this end we build certain edge sets. First, we create the chains of vertices in U which will be included in both layers of G'. Thus, for each edge $e_j \in E$ and each vertex $v_i \in e_j$ we build the following $(k' + 2)$-long path which we call (i, j)-*chain* or CH_j^i:

$$\mathrm{CH}_j^i := \bigcup_{h \in [k'+1]} \left\{\{j_h^i, j_{h+1}^i\}\right\}.$$

Note that there are two (i, j)-chains for each $j \in [m]$. Later, if we include such an (i, j)-chain to one path in a solution, then it shall also be included in the other path of that solution. For this reason the chain contains $k' + 1$ unique edges (from which later either all or none can be included in any st-path).

In the first layer each st-path will contain exactly one (i, j)-chain for each $j \in [m]$. The edge set E_1 for the first layer is then built as follows:

$$E_1 = \bigcup_{e_j \in E} \bigcup_{v_i \in e_j} \mathrm{CH}_j^i \cup \mathrm{ST} \cup \left\{\{s, X_1\}, \{X_{(m+1)}, t\}\right\}.$$

Here, ST is defined in the following way:

$$\mathrm{ST} := \bigcup_{j \in [m]} \bigcup_{v_i \in e_j} \left\{\{X_j, j_1^i\}, \{j_{(k'+2)}^i, X_{(j+1)}\}\right\}.$$

In the second layer of G' there shall be an st-path including all vertices in V. This main path then branches at each $v_i \in V$ to a "detour" where (i, j)-chains can be included to it (at the cost of taking the detour, which affects the edge difference). To this end, we first build the main path:

$$\mathrm{MA} := \{\{s, v_1\}, \{v_{n+1}, t\}\} \cup \bigcup_{i \in [n]} \{\{v_i, v_{i+1}\}\}.$$

Then we construct one detour path via vertices in Y and Z for each $i \in [n]$:

$$\mathrm{DET}_i := \{\{v_i, Y_1^i\}, \{Z_m^i, v_{i+1}\}\} \cup \bigcup_{j\in[m]} \{\{Y_j^i, Z_j^i\}\} \cup \bigcup_{j\in[m-1]} \{\{Z_j^i, Y_{j+1}^i\}\}.$$

Each existing (i,j)-chain is then "attached" to the corresponding Y_j^i and Z_j^i :

$$\mathrm{AT} := \bigcup_{e_j\in E} \bigcup_{v_i\in e_j} \{\{j_1^i, Y_j^i\}, \{j_{k'+2}^i, Z_j^i\}\}.$$

With this, the edge set E_2 of the second layer can be constructed:

$$E_2 = \mathrm{MA} \cup \mathrm{AT} \cup \bigcup_{e_j\in E} \bigcup_{v_i\in e_j} \mathrm{CH}_j^i \cup \bigcup_{i\in[n]} \mathrm{DET}_i.$$

We then finish building our temporal graph $G' = (V', E_1, E_2)$. The constructed input instance of Edge-MstP is (G', s, t, k', ℓ). ♦

An illustration of Construction 2.3 is shown in Figure 2.1.

In order to use Construction 2.3 to prove Theorem 2.1, we first need to make sure that this construction describes a correct reduction from VC to Edge-MstP:

Lemma 2.4. The constructed instance of Construction 2.3 is a YES-instance of Edge-MstP if and only if the input instance is a YES-instance of VC.

The proof of Lemma 2.4 is split into one lemma for the forward direction and one lemma for the backward direction.

Lemma 2.5. If the input to Construction 2.3 is a YES-instance of VC, the output is also a YES-instance of Edge-MstP.

Proof. Let $C \subseteq V$ be a set of size at most k in the input instance so that for each $e_j \in E$ holds $e_j \cap C \neq \emptyset$. We construct a solution $S = (P_1, P_2)$ to the output instance as follows:

For each $e_j \in E$ we pick one $v_i \in e_j \cap C$ and include the corresponding (i,j)-chain to P_1 and P_2. To P_1 we also include the vertices in $X \cup \{s,t\}$. With that we can see that P_1 is an st-path in $G_1' = (V', E_1)$ with $|V(P_1)| \leq \ell$.

Let $W^i = \{Y_x^i, Z_x^i : x \in [m]\}$ for each $v_i \in C$. Let W be the union of all W^i. Note that for each (i,j)-chain in P_2 both j_1^i and $j_{k'+2}^i$ have a distinct neighbour in W. Therefore we also include the vertices in $W \cup V \cup \{s,t\}$ to P_2. With this, P_2 is also an st-path of length at most ℓ (that "detours" at each $v_i \in C$ to contain all vertices from W^i and the chosen (i,j)-chains).

Next we have to ensure that the edge difference of P_1 and P_2 is at most k'. In both paths there are the same (i,j)-chains, so all edges included in these chains are not in

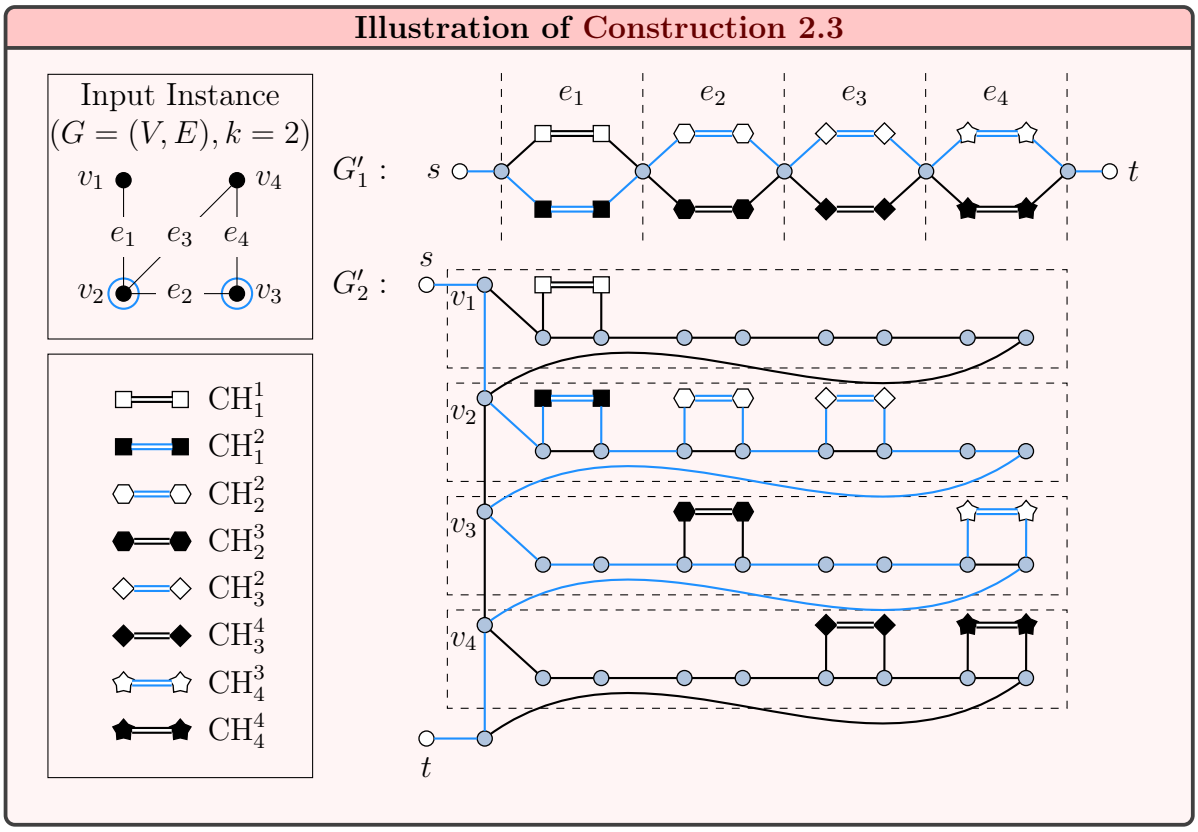


Figure 2.1: On the top left, an example instance $(G = (V, E), k)$ of Vertex Cover is shown. On the right, the temporal graph G' built by Construction 2.3 for this input instance is illustrated. This temporal graph has two layers, G'_1 and G'_2. The other output parameters for this input are $k' = (2k+3)|E|+|V|+4 = 36$ and $\ell = 2|E||V| + m(k'+2) + |V| + 42 = 230$. For each $e_j \in E$ and $v_i \in e_j$ we indicate the constructed $(k'+2)$-long chain CH_j^i by a double edge. All these chains are listed in the legend on the bottom left. Vertices which are coloured grey have degree zero in the other layer and thus are not depicted there. We indicate a possible solution $S = (P_1, P_2)$ for Edge-MstP of the output instance by drawing edges included in P_1 or P_2 blue, making both st-paths visible this way. Other edges are drawn black. The to S corresponding vertex cover vertices in the input instance are marked by a blue circle.

$E(P_1) \triangle E(P_2)$. In P_1 all other edges have endpoints in X. As $|X| = m + 1$, there can be at most $2m + 2$ edges present in P_1, but not present in P_2.

Let $V^* = V \cup \{v_{n+1}, s, t\}$. In $V(P_2)$ there are the vertices from V^*, the vertices from W and the vertices from $|E|$ many chains (from which the edges inside are both included in P_1 and P_2). As P_2 is a path, there can then be at most $|V^*|+|W|+|E|-1 = n+3+2km+m-1$ edges present in P_2 which are not present in P_1. Hence, the total edge difference is at most $2m + 2 + n + 3 + 2km + m - 1 = (2k + 3)m + n + 4 = k'$. Therefore S is a solution and the constructed instance is a YES-instance. □

Lemma 2.6. If the output instance of Construction 2.3 is a YES-instance of Edge-MstP, the input instance was a YES-instance of VC as well.

Before proving Lemma 2.6 we observe two different properties of any solution to Edge-MstP for an instance constructed by Construction 2.3:

Observation 2.7. If a solution $S = (P_1, P_2)$ for Edge-MstP to the instance constructed by Construction 2.3 exists, then

(a) for each $e_j \in E$ in the input instance there is exactly one constructed (i, j)-chain included in P_1 and

(b) P_2 includes the same (i, j)-chains as P_1.

Proof. (a) This follows from the construction of $\mathrm{ST} \subseteq E_1$, for each $e_j \in E$ there are exactly two constructed paths between X_j and X_{j+1}, each via one (i, j)-chain. Note that $\{X_j\}$ and $\{X_{j+1}\}$ both form st-separators, so P_1 includes X_j and X_{j+1}.

(b) Each (i, j)-chain contains $k' + 1$ edges. As the edge difference of P_1 and P_2 is at most k', they include the same such chains. □

With Observation 2.7 we now derive the following statement, which will later be used as our main argument in the proof of Lemma 2.6:

Lemma 2.8. If a solution $S = (P_1, P_2)$ to the instance constructed by Construction 2.3 exists, the set

$$I = \{i : \text{there is a } j \in [m] \text{ so that the } (i, j)\text{-chain is contained in } P_2\}$$

has at most size k, i.e. $I \leq k$ (k being the parameter of the input instance).

Proof. Assume towards a contradiction that there is a solution $S = (P_1, P_2)$ to the constructed instance so that $|I| \geq k + 1$. As the vertices in X are included only in P_1 and there are no two distinct $x, x' \in X$ with $\{x, x'\} \in E_1$, there are at least $2m + 2$ edges present in P_1 which are not present in P_2.

By Observation 2.7 we also know that there are m different (i,j)-chains in P_2, which respectively need at least m edges to get connected to other vertices in P_2. As $|I| \geq k+1$, we also have at least $(k+1)(2m)$ vertices from $Y \cup Z$ in $V(P_2)$ (which are not in $V(P_1)$). Then we need at least $(k+1)(2m)$ edges in $E(P_2)$ to connect each of them to other vertices as well. The same holds for the vertices in $V \cup \{v_{n+1}, t\}$ (not counting s, because it is the first vertex of the path). Altogether this means that there are at least $m + (k+1)2m + (n+2)$ edges included in P_2 which are not included in P_1.

Hence, the total edge difference of P_1 and P_2 is least $(2m+2) + m + (2k+2)m + (n+2) = (2k+5)m + n + 4 > k'$. This contradicts to S being a solution. □

We next prove Lemma 2.6 by showing that the set I of size at most k we found in Lemma 2.8 corresponds to a vertex cover of the same size in the input instance of Construction 2.3.

Proof of Lemma 2.6. If the output instance of Construction 2.3 is a YES-instance, let $S = (P_1, P_2)$ be a corresponding solution for Edge-MstP. By Observation 2.7 we know that both P_1 and P_2 contain exactly one (i,j)-chain for each $e_j \in E$. By Lemma 2.8 we know that the set

$$I = \{i : \text{there is a } j \in [m] \text{ so that the } (i,j)\text{-chain is contained in } P_2\}$$

is of size at most k. As we constructed an (i,j)-chain for the edge e_j if and only if $v_i \in e_j$, I translates to a set $C = \{v_i : i \in I\} \subseteq V$ with $|C| \leq k$, which is a vertex cover in the input graph. This means the input instance of Construction 2.3 was a YES-instance of VC. □

The proof of Lemma 2.4 is immediate by Lemma 2.5 together with Lemma 2.6.

In order to eventually prove Theorem 2.1 it remains to show that Construction 2.3 can be applied in time polynomial in the size of its input:

Observation 2.9. Construction 2.3 can be executed in $O((km+n)m)$ time.

Proof. Each constructed (i,j)-chain contains $k'+2 = (2k+3)m + n + 6$ vertices. As there are $2m$ such constructed chains these amount to $O((km+n)m)$ vertices in U. As $|X| = m+1$ and $|Y| = |Z| = nm$, there are $O((km+n)m)$ constructed vertices in total. For each constructed $v \in V'$ holds $\deg(v) \leq 4$ in each layer and $\tau = 2$. This means that there are also at most $O((km+n)m)$ constructed temporal edges. □

So far, the maximal degree of vertices generated by Construction 2.3 within any layer is four, which is not sufficient to prove Theorem 2.1. However, we can modify any output instance of Construction 2.3 to shrink this maximal degree down to three:

Proposition 2.10. Each instance I constructed by Construction 2.3 can be converted in polynomial time into an equivalent instance I' of Edge-MstP, so that the maximal degree of vertices in the underlying graph of the temporal graph in I' is 3.

Proof. For each $v \in U \cup Y \cup Z \cup \{X_1, X_{m+1}, v_1, v_{n+1}\}$ constructed by Construction 2.3 it already holds that $\deg_{G_\downarrow}(v) \leq 3$.

For each $X_j \in X \setminus \{X_1, X_{m+1}\}$ we observe that $\deg_{G_1}(X_j) = \deg_{G_1}(X_j) = 4$ and that all st-paths containing X_j go from a vertex in $L = \{(j-1)^i_{k'+2}, (j-1)^{i'}_{k'+2}\}$ via X_j to a vertex in $R = \{j^x_{k'+2}, j^{x'}_{k'+2}\}$ for some $i, i', x, x' \in [n]$ (also see Figure 2.1 for reference). Then we can exchange X_j for two vertices $X^*_j, {X'}_j$ of degree 3 in G_1 so that $\{X^*_j, {X'}_j\} \in E_1$, $\{X^*_j, w\} \in E_1$ for each $w \in L$ and $\{X'_j, u\} \in E_1$ for each $u \in R$. We also increase k' by the extra amount of vertices in each st-path in G_1, which is $m-1$ (as we exchanged $m-1$ vertices into two each). We also increase the length of all constructed (i,j)-chains accordingly.

The same idea is applied to the vertices in $V \setminus \{v_1, v_{n+1}\}$: Each $v_i \in V \setminus \{v_1, v_{n+1}\}$ has a degree of 4 in G_2 (and also in $G_\downarrow$). Each st-path through v_i comes from v_{i-1} or Z^{i-1}_m and goes via v_i to v_{i+1} or Y^i_1. This means we may also exchange v_i for two new vertices as depicted above. After all v_i have been exchanged, we increase k' by $n-1$ and again update all constructed (i,j)-chains accordingly. □

Now we are set to prove Theorem 2.1:

Proof of Theorem 2.1. As can be seen from the construction (or Figure 2.1) both layers constructed by Construction 2.3 are outerplanar graphs. This does not change when Proposition 2.10 is applied, as its vertex exchanges do not enclose other vertices and thus do not lead to any vertex being in a new graph face. Together with this observation, Lemma 2.4, Observation 2.9 and Proposition 2.10 directly prove Theorem 2.1. □

2.2 Polynomial Time Reduction from Edge-MstP to Vertex-MstP

In the previous section, we showed that Edge-MstP is NP-hard even if there are only two layers of treewidth two in the input temporal graph and the maximal degree in $G_\downarrow$ is three. We now show that those properties also hold for Vertex-MstP:

Theorem 2.11: NP-hardness of Vertex-MstP with only two layers of treewidth two

Vertex-MstP is NP-hard, even if the maximal degree in $G_\downarrow$ is 3, $\tau = 2$ and both input layers have a treewidth of at most two.

In order to prove this theorem, we first present an algorithm which describes a reduction from Edge-MstP to Vertex-MstP. We then show the correctness of this reduction in Lemma 2.12. After that, we can show the statements of Theorem 2.11 by proving that the presented algorithm does not change any properties listed in Theorem 2.11 from its input temporal graph to a constructed temporal graph.

Algorithm 1: A polynomial time reduction from Edge-MstP to Vertex-MstP

Input : An instance $I = (G = (V, E_1, \ldots, E_\tau), s, t, k, \ell)$ of Edge-MstP.
Output : An instance $I' = (G' = (V', E'_1, \ldots, E'_\tau), s, t, k', \ell')$ of Vertex-MstP which is equivalent to I.

$V' \leftarrow V$
for $i \leftarrow 1$ **to** τ **do**
  $E'_i \leftarrow E_i$
end
for $i \leftarrow 1$ **to** τ **do**
  foreach $e = \{a, b\} \in E_i$ **do** /* replacing e with a $k+1$-long path */
    $V' \leftarrow V' \cup \bigcup_{x \in [k+1]} \{e_x\}$
    $E'_i \leftarrow E'_i \cup \bigcup_{x \in [k]} \{\{e_x, e_{x+1}\}\} \cup \{\{a, e_1\}, \{b, e_{k+1}\}\}$
    $E'_i \leftarrow E'_i \setminus \{e\}$
  end
end
$k' \leftarrow (k+1)^2 - 1$ /* computing output parameters */
$\ell' \leftarrow \ell + (\ell - 1)(k+1)$
return $(G' = (V', E'_1, \ldots, E'_\tau), s, t, k', \ell')$

Intuitively, Algorithm 1 replaces each temporal edge of the temporal graph in the input instance (G, s, t, k, ℓ) with a path that includes $k+1$ new vertices, resulting in a new temporal graph G'. Also, the parameters k' and ℓ' of the output instance are computed as $k' = ((k+1)^2 - 1)$ and $\ell' = \ell + (\ell - 1)(k+1)$.

We now prove that each output instance I' of Algorithm 1 to Vertex-MstP is equivalent to the corresponding input instance I of Edge-MstP, i.e. that the reduction described by Algorithm 1 is correct:

Lemma 2.12. The output instance of Algorithm 1 is a yes-instance of Vertex-MstP if and only if the input instance is a yes-instance of Edge-MstP.

Proof. ($\Leftarrow$) Assume the input instance is a yes-instance. Let $S = (P_1, \ldots, P_\tau)$ be a solution to the input instance. We construct a solution $S' = (P'_1, \ldots, P'_\tau)$ to the output instance by replacing each edge $e = \{a, b\}$ in each P_i with the $k+3$-long path from a to b (via new vertices $\{e_1, \ldots, e_{k+1}\}$) that was inserted in Algorithm 1 for e.

As each P_i contains at most $\ell - 1$ edges and for each edge we inserted $k+1$ additional

vertices, the amount of vertices in P'_i is at most $\ell + (\ell - 1)(k + 1)$.

Now let P_i and P_{i+1} be two adjacent paths in S, which means their edge difference is at most k. This means the vertex difference of P_i and P_{i+1} is also at most k, since for each vertex $v \in V(P_1) \setminus \{s, t\}$ and $v \notin V(P_{i+1}) \setminus \{s, t\}$ there also has to be a distinct edge $e \in E(P_i)$, which is incident to v in P_i but not included in P_{i+1} and vice-versa. As $|E(P_i) \triangle E(P_{i+1})| \leq k$ there are at most $k(k+1)$ "new" vertices inserted by Algorithm 1 which are in $|V(P'_i) \triangle V(P'_{i+1})|$. Thus, the total vertex difference of P'_i and P'_{i+1} is at most $k + k(k + 1) = (k + 1) + k(k + 1) - 1 = (k + 1)^2 - 1 = k'$. Hence, S' is a solution to Vertex-MstP.

($\Rightarrow$) Assume the input instance is a no-instance. Then at least one of the following three cases applies:

1. There is a layer G_i in the input where no st-path exists.
2. There is a layer G_i in the input where each st-path contains more than ℓ vertices.
3. In each $S = (P_1, \ldots, P_\tau)$ where each P_i is an st-path of length at most ℓ in the input layer G_i there are two paths P_i and P_{i+1} so that $|E(P_i) \triangle E(P_{i+1})| \geq k + 1$.

Case 1: As Algorithm 1 only replaces edges that are present in E_i with longer paths in E'_i, it does not change connectivity of the vertices in V in any layer: If there is no path from s to t in G_i, then there also is no path from s to t in G'_i.

Case 2: Let P_i be an existing st-path in G_i with $|V(P_i)| > \ell$ (if P_i does not exist, Case 1 applies). Then P_i includes at least $\ell + 1$ vertices and at least ℓ edges. Thus, the path that emerges from P_i after replacing its included edges with the corresponding paths constructed in Algorithm 1 includes at least $\ell + 1 + \ell(k + 1)$ vertices and is consequently of length more than ℓ.

Case 3: Let P'_i and P'_{i+1} be the paths in G'_i that are constructed by exchanging the edges in P_i and P_{i+1} for the corresponding constructed $k + 1$-long paths. Then $|V(P'_i) \triangle V(P'_{i+1})| \geq (k + 1)(k + 1) = (k + 1)^2 > k'$ and there is no solution of Vertex-MstP to the output instance as a consequence. □

Having shown that Algorithm 1 is a correct reduction from Edge-MstP to Vertex-MstP, we are now set to prove Theorem 2.11.

Proof to Theorem 2.11. Edge-MstP is NP-hard even if $\tau = 2$, both layers are series-parallel and $G_\downarrow$ has a maximal degree of 3 (Theorem 2.1). Let I be an instance of Edge-MstP with only two series-parallel layers and a maximal degree of 3 in $G_\downarrow$. Then I can be reduced in $O(mk)$ time by Algorithm 1 to an equivalent instance I' of Vertex-MstP (Lemma 2.12) which also has those properties. This results from the fact that all modifications applied in Algorithm 1 are series expansions which do not introduce new vertices of degree greater than two. □

2.3 ETH Statement for Vertex-MstP

The Exponential Time Hypothesis (ETH) is a conjecture first suggested by Impagliazzo and Paturi (1999). It states that there is no algorithm which decides 3-SAT in $2^{o(n)}(n+m)^{O(1)}$ time, where 3-SAT is the problem to decide whether a given boolean formula in conjunctive normal form with m clauses, each containing three of n variables, can be satified.

In the previous section, we showed that there is no polynomial time algorithm for solving Vertex-MstP unless P = NP. Following this line of thought, we next show an even higher running time lower bound for solving Vertex-MstP by assuming the ETH:

Theorem 2.13: ETH Statement for Vertex-MstP

Unless the Exponential Time Hypothesis fails, Vertex-MstP cannot be solved in $2^{o(|V|)}(|V|+\tau)^{O(1)}$ time.

This section is thus primarily dedicated to proving Theorem 2.13. At its end, we also derive Corollary 2.20 which states that Vertex-MstP is NP-hard even if $k = 0$. In order to do so, we will build upon an ETH lower bound statement formulated by Muzi et al. (2017) for the following problem:

Definition 2.14: Positive 1-in-3-SAT

Input: A set of clauses $\{C_1, \ldots, C_m\}$, each consisting of exactly three out of n boolean variables $\{x_1, \ldots, x_n\}$.
Question: Is there an assignment to all the variables so that within each clause there is exactly one variable assigned to *true*?

We always assume that $m > 1$ and $n > 3$, otherwise the instance can be solved trivially. The Positive 1-in-3-SAT problem was shown to be NP-complete by Schaefer (1978). However, our main interest for it lies in the aforementioned theorem by Muzi et al. (2017):

Proposition 2.15. Unless the Exponential Time Hypothesis fails, Positive 1-in-3-SAT cannot be solved in $2^{o(n)}(n+m)^{O(1)}$ time.

We next describe a polynomial time reduction from Positive 1-in-3-SAT to Vertex-MstP which will "translate" each variable into a constructed vertex and each clause into a temporal graph layer. Each of those layers will then guarantee that exactly one of three vertices corresponding to variables of one clause can be included into an st-path. This way, we will later establish the correctness of the reduction Afterwards, we will use it together with Proposition 2.15 to prove Theorem 2.13.

Construction 2.16. Let $(\{C_1, \ldots, C_m\}, \{x_1, \ldots, x_n\})$ be an instance of Positive 1-in-3-SAT. Let $k = 0$. Let $\ell = n + 42$.

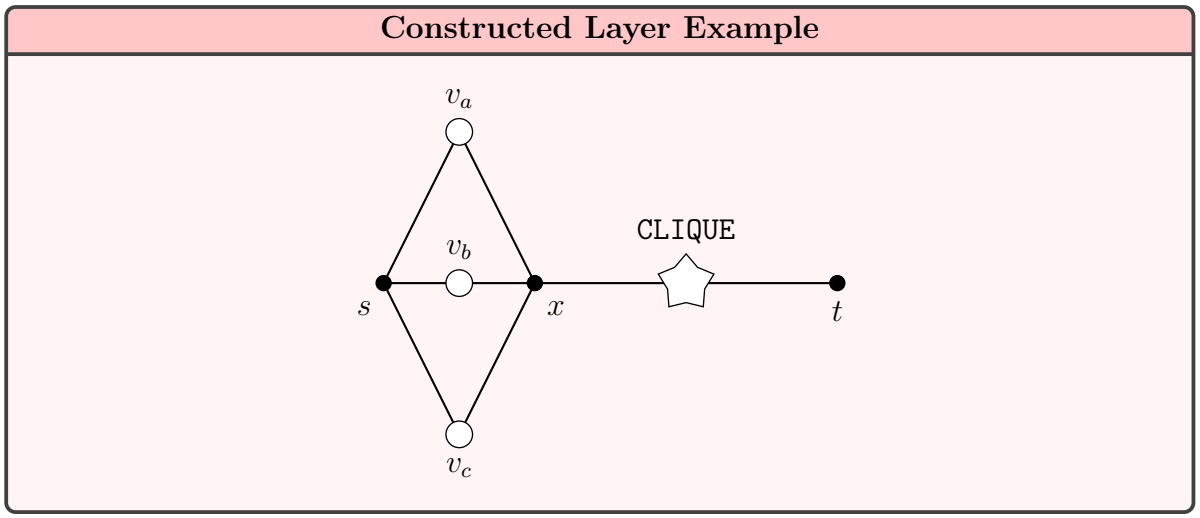


Figure 2.2: This picture represents one layer constructed by Construction 2.16 for any clause $\{x_a, x_b, x_c\}$. Note that `CLIQUE` refers to a complete subgraph formed by all vertices that are otherwise unused in that layer, that is, $V \setminus \{s, t, x, v_a, v_b, v_c\}$.

We construct a temporal graph $G = (V, E_1, \ldots, E_m)$ with

$$V = \{v_i \mid i \in [n]\} \cup \{s, t, x\}\,.$$

Later, adding v_i to a path in a solution shall represent the corresponding variable x_i to be assigned to *true*. Respectively, not adding v_i shall represent the corresponding variable x_i to be assigned to *false*.

For each clause $C_j = (x_a, x_b, x_c)$, $j \in [m]$, construct a layer G_j in which there are three parallel paths (between s and x), one via each vertex for a variable in C_j. The vertices which correspond to variables of other clauses are not important in G_j, but should still be "accessible". For this purpose, we make them form one big clique and let each of them be adjacent to both x and t. Formally, we construct the edge set E_j for C_j as follows. Let $Q = \{v_a, v_b, v_c\}$. Let $R = V \setminus \{v_a, v_b, v_c, s, x, t\}$. Let

$$E_j = \bigcup_{v_i \in Q} \Big\{\{s, v_i\}, \{x, v_i\}\Big\} \cup \bigcup_{u \in R} \Big\{\{t, u\}, \{x, u\}\Big\} \cup \bigcup_{u,w \in R} \Big\{\{u, w\}\Big\}.$$

This edge set is illustrated by an example in Figure 2.2. This finishes the construction, the output instance is (G, s, t, k, ℓ). ⧫

We next show that output instances of Construction 2.16 are YES-instances of Vertex-MstP if and only if the input instance is also a YES-instance of Positive 1-in-3-SAT. For this, we first note that in any solution to such a constructed instance the vertices of all paths are identical. Formally:

Observation 2.17. Let $(P_1, P_2, \ldots, P_\tau)$ be a solution to an output instance of Construction 2.16. Then $V(P_1) = V(P_2) = \cdots = V(P_\tau)$.

Proof. Let $S = (P_1, P_2, \ldots, P_\tau)$ be a solution to such an instance. Then by Construction 2.16 we know that the parameter k is set to zero. Hence $V(P_x) = V(P_{x+1})$ holds for all $x \in [\tau - 1]$, because otherwise $|V(P_x) \triangle V(P_{x+1})| > k = 0$, which would contradict to S being a solution. Now Observation 2.17 follows directly. □

As stated earlier, the vertices in paths of a solution for the constructed instance shall represent which variables should be assigned to *true* in a solution for the input instance. Using this one-to-one mapping between *true*-assigned variables in the input and vertices included in paths of a solution for the output, we will now show the first direction of the reduction correctness:

Lemma 2.18. If the constructed instance of Construction 2.16 is a YES-instance of Vertex-MstP, then the input instance was also a YES-instance of Positive 1-in-3-SAT.

Proof. Let S be a solution to the corresponding constructed instance. Let $C_j = \{x_a, x_b, x_c\}$ be a clause in an input instance of Construction 2.16. Then there is a layer G_j in the constructed instance that was created for C_j. In that layer every st-path includes exactly one of the vertices $\{v_a, v_b, v_c\}$ (because each path between s and x includes one of them and $\{x\}$ is an st-separator). Hence, there is also a path P_j in S including exactly one of the vertices $\{v_a, v_b, v_c\}$. Now let P be any path in S. By Observation 2.17 it holds that $V(P) = V(P_j)$, which means especially that P also includes exactly one of the vertices $\{v_a, v_b, v_c\}$. Since C_j was chosen arbitrarily, it follows that the vertices in $P \setminus \{s, x, t\}$ correspond to a variable assignment which leads to a solution for the input instance (as all clauses can be "satisfied" by that assignment). □

Now we know that if a solution to the constructed instance exists, then there is also a solution to the input instance. The other direction works similarly:

Lemma 2.19. If the input instance of Construction 2.16 is a YES-instance of Positive 1-in-3-SAT, then the constructed instance is a YES-instance of Vertex-MstP as well.

Proof. As there is a solution to the input instance, let X be the set of all variables assigned to *true* in that solution. Let $W = \{v_i : x_i \in X\} \cup \{s, x, t\}$ be the corresponding subset of V in the constructed instance. Let G_i be any layer in that constructed instance. As G_i was constructed for a particular input clause $C_i = \{x_a, x_b, x_c\}$, exactly one of the three vertices $\{v_a, v_b, v_c\}$ is in W and there is an sx-path in G_i via this

vertex. The vertices in $W \setminus \{v_a, v_b, v_c\}$ are in a clique in G_i, so any combination out of them forms an xt-path. Thus, there is an st-path in G_i which is formed by exactly the vertices in $W \cup \{s, x, t\}$. □

Now we are set to prove Theorem 2.13:

Proof of Theorem 2.13. By Lemmata 2.18 and 2.19 we know that the instance constructed by Construction 2.16 is equivalent to its input instance. It is also evident that Construction 2.16 takes polynomial time in its input size, since there is one constructed layer per input clause and each constructed layer contains at most n^2 temporal edges.

This polynomial-time reduction from Positive 1-in-3-SAT to Vertex-MstP together with Proposition 2.15 then directly proves Theorem 2.13, as for the output instance of Construction 2.16 it holds that $|V| = n + 3$ and $\tau = m$. □

As Positive 1-in-3-SAT is NP-complete and Construction 2.16 is a polynomial-time reduction from Positive 1-in-3-SAT to Vertex-MstP which always sets the output parameter k to zero we also derive the following corollary:

Corollary 2.20: NP-Hardness of Vertex-MstP with $k = 0$

Vertex-MstP is NP-hard, even if $k = 0$.

3 Parameterised Hardness

In the following chapter, we study Multistage st-Path as a parameterised problem. We first establish that all researched problem variants are W[1]-hard when parameterised by τ, k and ℓ combined (Section 3.1). In Section 3.2, we show W[1]-hardness of problem variants Vertex-MstP and Edge-MstP with respect to the vertex cover number of the underlying graph of the input temporal graph.

3.1 W[1]-Hardness for Input Parameters τ, k and ℓ

In this section, we study the parameterised complexity of Multistage st-Path with respect to the three most natural parameters in any input instance, that being the number of layers τ, the maximal length of st-paths ℓ, and the maximal distance between adjacent st-paths in a solution k. We already showed in Theorems 2.1 and 2.11 that Vertex-MstP and Edge-MstP are both NP-hard when τ is two. Hence, there can be no FPT-time algorithm for those two problems with respect to only the parameter τ (unless P = NP). For this reason we now combine the three parameters, i.e. we investigate if there is an FPT-time algorithm for our researched variants of Multistage st-Path when parameterised by $\tau+k+\ell$. Unfortunately the answer is still negative for all variants, unless FPT = W[1].

In order to prove this claim, this section is divided in two parts: At first, we show that Vertex-MstP parameterised by $\tau + k + \ell$ is W[1]-hard (Theorem 3.1). In the second part, we derive similar W[1]-hardness results for Edge-MstP and Leven-MstP (Corollary 3.8).

3.1.1 W[1]-Hardness of Vertex-MstP for τ, k, and ℓ

In this subsection, we prove the following:

Theorem 3.1: W[1]-Hardness of Vertex-MstP with τ, k, and ℓ as Parameters

Vertex-MstP is W[1]-hard when parameterised by $\tau + \ell$, even if $k = 0$ and all layers in the input are bipartite graphs.

For this purpose we will describe a polynomial parameter transformation to Vertex-MstP from Multicoloured Clique. This problem is W[1]-complete when parameterised by its colour number parameter k (Fellows et al. 2009).

Definition 3.2: Multicoloured Clique

Input: An undirected graph $G = (V, E)$, an integer k, and a function $c : V \to [k]$.
Question: Is there a subset $\mathcal{C} \subseteq V$ such that $\mathcal{C}$ forms a clique of size k and $\bigcup_{v \in \mathcal{C}} c(v) = [k]$?

Intuitively we interpret the function c as a colouring of vertices, that is, each vertex v has the colour $c(v)$. Furthermore we will refer to a set $S \subseteq V$ as a multicoloured clique if and only if the induced subgraph of S in G is a clique and there are no two distinct vertices $v, w \in S$ so that $c(v) = c(w)$. We also always assume that $3 \leq k \leq |V|$ and that there is at least one vertex $v \in V$ of each colour $i \in [k]$. Otherwise deciding whether the instance is a yes-instance of Multicoloured Clique would be trivial.

Construction 3.3. Pseudo-code of the following construction is stated in Algorithm 2.

Let $(G = (V, E), k, c)$ be an instance of Multicoloured Clique. Let $\ell = k + 2$ and $k' = 0$. These two parameters will be used in the output instance, note that both are linear in k. We construct a temporal graph G' with vertex set $V' = V \cup \{s, t\}$ and edge sets $E_1, \ldots, E_{1+k(k-1)}$.

For each of the edge sets in G' we arrange the k colours in a specific order, i.e. we use $k(k-1)+1$ permutations of colours. The constructed edge sets will later each reflect their corresponding colour order. To this end, we begin with the natural colour order $(1, \ldots, k)$ (Algorithm 2 line 3). Afterwards for each $i \in [k]$ we "shift" colour i to the right $k-1$ times, i.e. for each $j \in [k-1]$ we swap colour i with colour $i + j \mod k$. Each such shift creates one new colour order (Algorithm 2 lines 4 to 10). Hence, we create $1 + k(k-1)$ colour orders in total, each one ordering all k colours in a specific way. By doing so, we ensure that each two distinct colours i, i' are ordered directly next to each other in at least one created order (as i was shifted $k-1$ times and after each shift it is ordered next to a new colour).

Next we build an edge set for each colour order. Thus let $\text{ORD}_i = (c_1, \ldots, c_k)$ be the i-th colour order. In edge set E_i, we add an edge $\{s, v\}$ for each $v \in V$ where $c(v)$ is the *first*-ordered colour c_1 in ORD_i. Similarly, we add an edge $\{t, v\}$ to E_i for each $v \in V$ where $c(v)$ is the *last*-ordered colour c_k in ORD_i (Algorithm 2 lines 13 and 14). Afterwards, we add an edge $\{a, b\}$ to E_i for each two $a, b \in V$ where $\{a, b\} \in E$ and $c(a) = c_j$ is ordered directly next to $c(b) = c_{j+1}$ in ORD_i, for $j \in [k-1]$ (Algorithm 2 lines 15 to 14).

This finishes the construction, the output instance is $(G' = (V', E_1, \ldots, E_\tau), s, t, k', \ell)$. An example output layer is illustrated in Figure 3.1. ♦

Construction 3.3 will later be used as polynomial parameter transformation in order to prove Theorem 3.1. For this reason, we prove that Construction 3.3 is indeed an appropriate polynomial parameter transformation from Multicoloured Clique to Vertex-MstP and make the following observation:

Algorithm 2: Algorithm for Construction 3.3

Input : A graph $G = (V, E)$, an integer $k \geq 3$, and a function $c : V \to [k]$.
Output : A temporal graph $G' = (V', E_1, \ldots, E_\tau)$, two special vertices $s, t \in V$, and two integers $k', \ell \in \mathbb{N}$.

$V' \leftarrow \{V \cup \{s, t\}\}$
/* Creating $k(k-1)+1$ permutations of the k colours */
cur $\leftarrow 1$
$\text{ORD}_{\text{cur}} \leftarrow (1, \ldots, k)$
for $i \leftarrow 1$ **to** k **do**
  for $j \leftarrow 1$ **to** $k-1$ **do**
    cur $\leftarrow$ cur $+ 1$
    $\text{ORD}_{\text{cur}} \leftarrow \text{ORD}_{\text{cur}-1}$
    Shift i one position to the right in ORD_{cur}
  end
end
/* Using each created permutation to build an edge set */
for cur $\leftarrow 1$ **to** $k(k-1)+1$ **do**
  $(c_1, \ldots, c_k) \leftarrow \text{ORD}_{\text{cur}}$
  $E_{\text{cur}} \leftarrow \{\{s, v\} : v \in V, c(v) = c_1\}$
  $E_{\text{cur}} \leftarrow E_{\text{cur}} \cup \{\{v, t\} : v \in V, c(v) = c_k\}$
  foreach $\{a, b\} \in E$ **do**
    if *there is an* $i \in [k-1]$ *so that* $c(a) = c_i$ *and* $c(b) = c_{i+1}$ **then**
      $E_{\text{cur}} \leftarrow E_{\text{cur}} \cup \{\{a, b\}\}$
    end
  end
end
/* Creating the output instance */
$G' \leftarrow (V', E_1, \ldots, E_{k(k-1)+1})$
$k' \leftarrow 0$
$\ell \leftarrow k + 2$
return (G', s, t, k', ℓ)

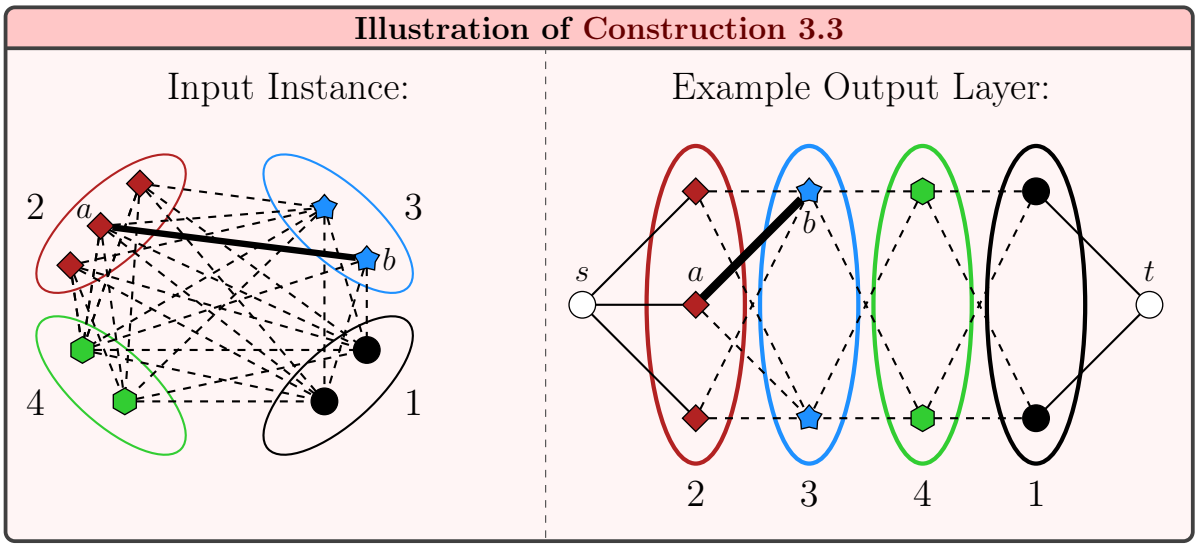


Figure 3.1: On the left there is a schematic input instance of Multicoloured Clique with 4 colours. Vertices are dyed and shaped according to their respective colour. On the right there is a corresponding constructed layer where the current colour order is $(2, 3, 4, 1)$. The edges indicated by dashed lines may exist (but do not have to), whereas $\{a, b\}$ represents an edge that does exist. Then $\{a, b\}$ is in the output layer, because the colour of a (which is 2) is ordered next to the colour of b (which is 3). Each dashed edge on the right side exists if the same edge also exists on the left side.

Observation 3.4. If $S = (P_1, P_2, \ldots, P_\tau)$ is a solution to the instance constructed by Construction 3.3, then $V(P_1) = V(P_2) = \cdots = V(P_\tau)$.

Proof. As k' is set to zero in each output instance of Construction 3.3, this statement follows directly. □

Next we need to show the reduction correctness:

Lemma 3.5. The output instance of Construction 3.3 is a YES-instance of Vertex-MstP if and only if the input is a YES-instance of Multicoloured Clique.

Proof. ($\Rightarrow$) If there is a multicoloured clique C in G of size k, all $v \in C \cup \{s, t\}$ form an st-path in each constructed layer. This results from the notion that for any order of the k colours, a path within C can be found that traverses the colours (or to be specifically the vertices with those colours) in that order. As these st-paths all include the same vertices, the output instance (G', s, t, k', ℓ) is a YES-instance of Vertex-MstP.

($\Leftarrow$) Assume that the output instance is a YES-instance of Vertex-MstP. Let $S = (P_1, \ldots, P_\tau)$ be its solution. Then P_1 contains a vertex of each colour, because $\{v : c(v) = i\}$ is a constructed st-separator for each $i \in [k]$. Let $C = V(P_1) \setminus \{s, t\}$. Note that $|C| = k$. Let $a, b \in C$ be two distinct vertices in C. Let $G_i = (V', E_i)$ be a layer that was constructed for a colour order where $c(a)$ is ordered next to $c(b)$. Such an order exists, as in Construction 3.3 it is ensured that each colour is ordered next to each other in at least one created colour order. We know that there is an st-path P_i with $V(P_i) = V(P_1) = C \cup \{s, t\}$ contained in G_i, since this path is in S. As P_i includes especially a and b, whose colours are ordered next to each other in the colour order for G_i, there has to be an edge $\{a, b\}$ in G_i. This means $\{a, b\}$ also exists in G, otherwise it would not have been constructed. Since a and b were chosen as any two distinct vertices in C, there is a clique in G formed by C, which is also of size k and multicoloured. Thus the input instance was a YES-instance of Multicoloured Clique. □

Lemma 3.6. Construction 3.3 is a polynomial parameter transformation from Multicoloured Clique parameterised by k to Vertex-MstP parameterised by $\tau + \ell$.

Proof. Construction 3.3 assigns the parameter $\tau + \ell$ polynomially upper-bounded in k, as $\tau + \ell + k' = (k(k-1)+1) + (k+2) = k^2 + 3$. It also can be executed in time $O((|V| + |E|)k^3)$, where $|V|$, $|E|$ and k are the respective parameters of the input instance. Together with Lemma 3.5 this proves the statement. □

We just established that Construction 3.3 is a polynomial parameter transformation

from a W[1]-complete problem to Vertex-MstP parameterised by $\tau+\ell$. In order to eventually prove Theorem 3.1, it now remains to show that all layers built by Construction 3.3 are bipartite:

Observation 3.7. Each temporal graph layer constructed by Construction 3.3 is a bipartite graph.

Proof. Let $G_i = (V', E_i)$ be a layer that was constructed for a particular colour order $\text{ORD} = (c_1, \ldots, c_k)$. Let $P : [k] \to [k]$ be the function that gives the index of a colour in ORD, i.e. $P(i) = j$ so that $c_j = i$ for any colour $i \in [k]$. Let $L = \{v \in V : P(c(v))$ is an odd number$\}$ and $R = \{v \in V : P(c(v))$ is an even number$\}$. Edges in G_i between two distinct vertices $a, b \in V$ only exist if their colours are ordered next to each other, i.e. if $|P(c(a)) - P(c(b))| = 1$. Therefore there is no $\{a, b\} \in E_i$ so that $a \in L$ and $b \in R$. Also s is only adjacent to vertices of colour c_1 in G_i and t is only adjacent to vertices of colour c_k in G_i. If k is even, then $(L \cup \{t\}, R \cup \{s\})$ is consequently a bipartition of G_i. If k is odd, then $(L, R \cup \{s, t\})$ is a bipartition of G_i instead. □

Putting the last two statements together, Theorem 3.1 follows directly:

Proof to Theorem 3.1. Construction 3.3 is a polynomial parameter transformation from the W[1]-hard Multicoloured Clique parameterised by k to Vertex-MstP parameterised by $\tau + \ell$ (Lemma 3.6). Moreover, in this construction the parameter k' is always assigned to zero and each layer is a bipartite graph by Observation 3.7. Thus, Vertex-MstP is W[1]-hard, even if $k = 0$ and each layer is a bipartite graph. □

3.1.2 W[1]-Hardness of Edge-MstP and Leven-MstP for τ, k, and ℓ

In the previous subsection we proved Vertex-MstP to be W[1]-hard for the combined parameter $\tau + k + \ell$ by describing a polynomial parameter transformation from Multicoloured Clique in Construction 3.3. Now we slightly modify this construction two times in order to show similar W[1]-hardness statements for Edge-MstP and Leven-MstP:

Corollary 3.8: W[1]-Hardness of Edge-MstP and Leven-MstP with τ, k, and ℓ as Parameters

The following problems are W[1]-hard when parameterised by $\tau + \ell$:

(1) Edge-MstP, even if $k = 4$ and all layers in the input are bipartite graphs.
(2) Leven-MstP, even if $k = 2$ and all layers in the input are bipartite graphs.

Proof. (1) We use Construction 3.3 as described to now output an instance of Edge-MstP, except for the parameter k', which gets changed from zero to four (meaning we change zero to four in line 22 of Algorithm 2). As a result, Observation 3.4 still holds, but its proof has to be adapted for this output instance of Edge-MstP with $k' = \ell$: We know that the used colour orders in Construction 3.3 always "swap" two colours c_j, c_{j+1} which are ordered next to each other from one created order to the next one (see Algorithm 2 line 7 and 8 in particular). Then, by extension, in each two adjacent constructed layers G_i, G_{i+1} the corresponding st-paths P_i, P_{i+1} each include two edges not included in the other one, namely an edge to a vertex of colour c_{j-1} and an edge to a vertex of colour c_{j+2}. These total to four edges in $E(P_i) \triangle E(P_{i+1})$, even if $V(P_i) = V(P_{i+1})$. If $V(P_i) \neq V(P_{i+1})$, i.e. if there was some vertex v included in P_i and not in P_{i+1} or vice-versa, then this vertex v would also lead to an additional edge included in one of those paths, of which v is an endpoint. Hence, we conclude that the edge difference of P_i and P_{i+1} is four if and only if $V(P_i) = V(P_{i+1})$. This not only proves the statement of Observation 3.4, but also implies that a solution for Edge-MstP can indeed be constructed in the same way as in the proof of Lemma 3.5. Thus, the remaining parts of proving this statement of Corollary 3.8 above are the same as for proving Theorem 3.1 earlier.

(2) The same arguments as for (1) apply, except that the parameter k' is now changed to two in line 22 of Algorithm 2 in order to output an instance of Leven-MstP. The Levenstein distance of P_i and P_{i+1}, which are defined as in the proof of (1) above, is then two, since the vertex string of P_{i+1} can be derived from the vertex string of P_i by two single vertex substitutions. □

As we did not show NP-hardness of Edge-MstP and Leven-MstP earlier for constant k and Construction 3.3 can be used as a polynomial-time reduction from the NP-complete Multicoloured Clique if modified as stated above, we additionally derive the following:

Corollary 3.9: NP-Hardness of Edge-MstP and Leven-MstP with constant k

The following problems are NP-hard:

(1) Edge-MstP, even if $k = 4$ and all layers in the input are bipartite graphs.
(2) Leven-MstP, even if $k = 2$ and all layers in the input are bipartite graphs.

Proof. For (1) we use Construction 3.3 as described, except that we change the parameter k' from zero to four (in line 22 of Algorithm 2). For (2) we change that parameter k' in Construction 3.3 to two instead. Then, as already explained in the proof of Corollary 3.8, the output instance of the respective problem (Edge-MstP in (1), Leven-MstP in (2)) is equivalent to the input instance of Multicoloured Clique. As Construction 3.3 takes time polynomial on its input size (Lemma 3.6) and each of its output layers is a bipartite graph (Observation 3.7), this proves both statements. □

3.2 W[1]-Hardness with Vertex Cover Number as Parameter

Fellows et al. (2008) and Fiala, Golovach, and Kratochvíl (2011), among others, suggested to research whether computational problems on simple graphs are in FPT when parameterised by the vertex cover number. They also showed that some problems are in FPT when parameterised that way but are W[1]-hard when parameterised by treewidth. Therefore, we investigated if Vertex-MstP and Edge-MstP are fixed-parameter-tractable when parameterised by the vertex cover number of the underlying graph of the input temporal graph. Unfortunately, we will show that both problems are intractable even if that vertex cover number is small (unless W[1] = FPT). In order to do so, we will present two polynomial parameter transformations within the next two subsections, one to Vertex-MstP and one to Edge-MstP.

3.2.1 W[1]-Hardness of Vertex-MstP parameterised by Vertex Cover Number

This subsection is dedicated to prove the following:

Theorem 3.10: W[1]-Hardness of Vertex-MstP with Vertex Cover Number, k, and ℓ as Parameters

Let ν be the vertex cover number of the underlying graph $G_{\downarrow}$ of the temporal graph in the input instance. Vertex-MstP is W[1]-hard when parameterised by $\nu + \ell$, even if $k = 1$.

In order to prove this theorem, we next describe a polynomial parameter transformation from the W[1]-complete Multicoloured Clique problem (as described in Definition 3.2) to Vertex-MstP:

Construction 3.11. Let $(G = (V, E), k, c)$ be an instance of Multicoloured Clique. Let $n = |V|$. Let $k' = 1$ and $\ell = 2k + 4$. The parameters k' and ℓ will be used in the output instance, note that both are linear in k. Let $H = [k + 1]$. We construct a temporal graph G' with vertex set $V' = V \cup \{s, t, X, Y\} \cup H$ and edge sets $E_1, \ldots, E_{2n(k-1)}$.

Each odd layer G_i shall ensure that there is exactly one vertex of each colour in any st-path in G_i. Moreover, all odd layers of G' are identical, i.e. $E_1 = E_3 = \cdots = E_{2n(k-1)-1}$. Let

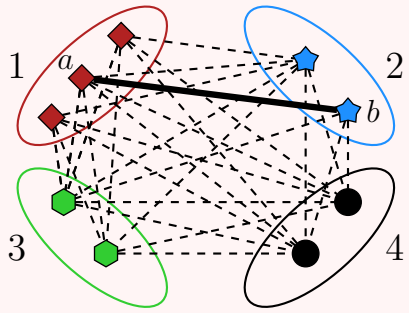
$$E' := \{\{s, 1\}, \{t, k + 1\}\} \cup \bigcup_{d \in [k]} \bigcup_{\substack{w \in V \\ c(w) = d}} \{\{d, w\}, \{w, d + 1\}\}$$

and let $E_i = E'$ for each odd $i \in [2n(k - 1)]$.

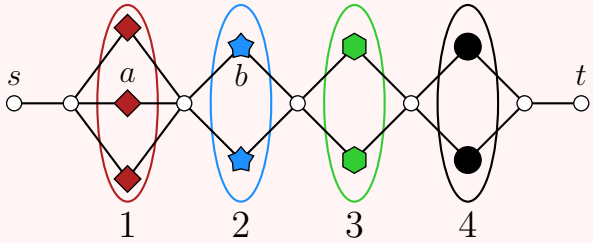
Next there shall be one even layer for each $v \in V$ and $j \in [k] \setminus \{c(v)\}$, totalling to $n(k - 1)$ even layers in G'. Thus construct an edge set E^v_j for each $v \in V$ and $j \in [k] \setminus \{c(v)\}$. Later, if v is included in an st-path P in the resulting layer, then P shall

Schematic of Construction 3.11

Input Instance:



Odd Output Layer:



Even Output Layer for Vertex a and Colour 2 :

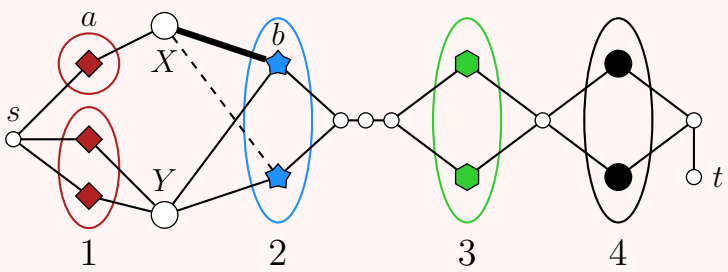


Figure 3.2: In this picture, examples of an odd and an even output layer for a small input instance are illustrated. The vertices in the minimum vertex cover are coloured white, whereas the vertices in V are dyed and shaped according to their respective colour. The edges indicated by dashed lines in the input may exist (but do not have to), whereas the thick line represents an edge that does exist. The illustrated even output layer is built specifically for vertex a and colour $c(b) = 2$. It contains the edge $\{X, b\}$, because in the input the edge $\{a, b\}$ exists (indicated by a thick line). It may also contain an edge $\{X, y\}$ for any $y \in V$ if $c(y) = 2$ and an edge $\{a, y\}$ is in the input graph (indicated by a dashed line).

also include a vertex $u \in V$ with colour j so that $\{u, v\} \in E$. To this end, E_j^v contains the following edge set:

$$\mathrm{REQ}_j^v := \{\{s, v\}, \{v, X\}\} \cup \bigcup_{\substack{w \in V \setminus \{v\} \\ c(w) = c(v)}} \{\{s, w\}, \{w, Y\}\} \cup \bigcup_{\substack{w \in V \\ c(w) = j}} \{\{Y, w\}, \{w, 1\}\} \cup \bigcup_{\substack{w \in V \\ c(w) = j \\ \{v,w\} \in E}} \{\{X, w\}\}.$$

Vertices of other colours than $c(v)$ and j are not important in that layer, but they nevertheless should be allowed to be included in an st-path. Hence, an edge set similar to the one in the odd layers is used for those vertices:

$$\mathrm{ST}_j^v := \{\{c(v), c(v)+1\}, \{j, j+1\}, \{t, k+1\}\} \cup \bigcup_{\substack{d \in [k] \\ d \neq j \\ d \neq c(v)}} \bigcup_{\substack{w \in V \\ c(w) = d}} \{\{d, w\}, \{w, d+1\}\}.$$

The edges $\{\{c(v), c(v)+1\}, \{j, j+1\}\}$ were added in order to complete st-paths. Then let $E_j^v = \mathrm{REQ}_j^v \cup \mathrm{ST}_j^v$. Each E_j^v is used as the edge set for a different even layer in G' (their order is irrelevant). This finishes the construction, the output instance is (G', s, t, k', ℓ). An example of this construction is illustrated in Figure 3.2. ♦

In order to show the correctness of Construction 3.11, we first observe the following property of solutions for the output instance:

Observation 3.12. If $S = (P_1, \ldots, P_\tau)$ is a solution to Vertex-MstP for an output instance of Construction 3.11, then $V(P_i) \setminus \{X, Y\} = V(P_{i+1}) \setminus \{X, Y\}$ for each $i \in [\tau - 1]$.

Proof. Let $i \in [\tau - 1]$. Let G_i and G_{i+1} be the i-th and $(i+1)$-th layer in the constructed temporal graph G'. Clearly, one of them is even and the other is odd. Without loss of generality, assume that G_i is the odd and G_{i+1} is the even layer. Vertices X and Y both have a degree of zero in each odd layer, so P_i includes neither X nor Y. In contrast to this, $\{X, Y\}$ is an st-separator in each even layer, so P_{i+1} includes X or Y. Thus, it holds that $X \in V(P_i) \triangle V(P_{i+1})$ or $Y \in V(P_i) \triangle V(P_{i+1})$ and hence $|V(P_i) \triangle V(P_{i+1})| \geq 1 = k'$. If there were more vertices than one out of $\{X, Y\}$ in $V(P_i) \triangle V(P_{i+1})$, then the vertex difference of P_i and P_{i+1} would consequently exceed $1 = k'$, contradicting to S being a solution. □

With the knowledge from Observation 3.12, we can now prove the correctness of Construction 3.11:

Lemma 3.13. The output of Construction 3.11 is a YES-instance of VERTEX-MSTP if and only if its input is a YES-instance of MULTICOLOURED CLIQUE.

Proof. ($\Leftarrow$) If there is a multicoloured clique C of size k in the input graph $G = (V, E)$, then let $R = C \cup H \cup \{s, t\}$. In each constructed odd layer G_i, R clearly forms an st-path P_i with $|V(P_i)| \leq \ell$ (as R contains exactly one vertex of each colour).

Now let G'_i be an even layer which is constructed specifically for a vertex $v \in V$ and a colour $j \in [k] \setminus \{c(v)\}$. Let u be the vertex in R which has the colour $c(v)$. If $u \neq v$, then there is an edge $\{u, Y\}$ in G'_i and $R \cup \{Y\}$ forms an st-path P'_i in G'_i (for this also see the constructed edge set ST^v_j in G'_i). Otherwise ($u = v$), there is an edge $\{v, X\}$ in G'_i. As C is a multicoloured clique there is a neighbour $w \in C$ of v in the input graph G which has colour j, meaning there is also an edge $\{w, X\}$ in G'_i. Then $R \cup \{X\}$ forms the st-path P'_i in G'_i instead of $R \cup \{Y\}$ in the this case (again, also see ST^v_j in G'_i). In both cases, it holds that $|V(P'_i)| \leq \ell$ and the edge difference of P_i and P'_i is at most $1 = k'$ (since either $P_i \triangle P'_i = \{X\}$ or $P_i \triangle P'_i = \{Y\}$). Thus, there is a solution to VERTEX-MSTP for the output instance which contains those paths.

($\Rightarrow$) If the input is a NO-instance of MULTICOLOURED CLIQUE, assume towards a contradiction that an output was a YES-instance of VERTEX-MSTP. Then there is a corresponding solution $S = (P_1, \ldots, P_\tau)$ for that output instance. Let $C = V(P_1) \cap V$. We know that C consists of exactly one vertex of each colour, otherwise P_1 could be no st-path in the first constructed layer. However, C forms no MULTICOLOURED CLIQUE in the input graph $G = (V, E)$ (as the input is a NO-instance). Then there are two distinct vertices $a, b \in C$ so that $\{a, b\} \notin E$.

Let G_i be the even layer constructed specifically for vertex a and colour $c(b)$. In this layer, $\{X, Y\}$ is an ab-separator. From Observation 3.12 we infer that $C = V(P_i) \cap V$, as $C = V(P_1) \cap V$. As P_{i-1} does not include X or Y (both have a degree of zero in the odd layer G_{i-1}), P_i does not include both X and Y or $|V(P_{i-1}) \triangle V(P_i)| > 1 = k'$ (which would contradict to S being a solution). Also there is no edge $\{a, Y\}$ in G_i, and since $\{a, b\} \notin E$, there is no edge $\{X, b\}$ in G_i either. Consequently P_i can not include a, b, and exactly one of X or Y, which contradicts to $C = V(P_i) \cap V$. □

As we want to establish that Construction 3.11 is a polynomial parameter transformation, it is still necessary to note that it has polynomial running time:

Observation 3.14. Construction 3.11 can be applied in $O(n^3)$ time, where n is the number of vertices in the input instance.

Proof. The number of constructed vertices is at most $2n + 5$, since the number of colours is at most n. In each output layer, each $v \in V$ is an endpoint of at most 3 edges and there are at most 3 edges per layer which do not have an endpoint in V. Thus, the number of edges per layer is at most $3n + 3$ and constructing each layer takes time in $O(n)$. There are at most $2n(n-1)$ constructed layers, leading to a total

running time in $O(n^3)$. □

The only remaining requirement for Construction 3.11 being a polynomial parameter transformation is that the "new" parameter (which is $\nu + k' + \ell$ in Vertex-MstP) is polynomially upper-bounded in the "old" parameter (which is the number of colours k in Multicoloured Clique). In the following argumentation, we will show that it does and with that prove our claim that Construction 3.11 is our desired polynomial parameter transformation:

Lemma 3.15. Construction 3.11 is a polynomial parameter transformation from Multicoloured Clique parameterised by k to Vertex-MstP parameterised by $\nu + k' + \ell$, where k is the number of colours in Multicoloured Clique, k' is the vertex difference parameter of Vertex-MstP, ℓ is the path length parameter of Vertex-MstP, and ν is the vertex cover number of the underlying graph of the temporal graph in instances of Vertex-MstP.

Proof. In each layer constructed by Construction 3.11 the vertex set $W = H \cup \{s, t, X, Y\}$ is a vertex cover with $|W| = k+5$, as each constructed edge has an endpoint in W. Thus, W also is a vertex cover of the underlying graph $G_{\downarrow}$. The parameters k' and ℓ are set to one and $2k + 4$, respectively. Therefore $\nu + k' + \ell$ is at most $(k + 5) + 1 + (2k + 4) = 3k + 10$. Together with Lemma 3.13 and Observation 3.14 it follows that Construction 3.11 is a polynomial parameter transformation from Multicoloured Clique parameterised by k to Vertex-MstP parameterised by $\nu + k' + \ell$. □

Having found this polynomial parameter transformation, the main theorem of this subsection now follows directly:

Proof of Theorem 3.10. As there is a polynomial parameter transformation from the W[1]-hard Multicoloured Clique parameterised by k (number of colours) to Vertex-MstP parameterised by $\nu + \ell + k$ (k being the maximal vertex difference here), the latter is also W[1]-hard. □

3.2.2 W[1]-Hardness of Edge-MstP parameterised by Vertex Cover Number

In the previous subsection we showed that Vertex-MstP is W[1]-hard when parameterised by the vertex cover number of the underlying graph of the temporal graph in the input (Theorem 3.1). This immediately poses the question if this W[1]-hardness also holds for Edge-MstP. In order to show that it does, we describe another polynomial parameter transformation from Multicoloured Clique which shares its basic ideas with Construction 3.11, but this time an instance of Edge-MstP is generated. Hence, in this subsection we prove the following theorem:

Theorem 3.16: W[1]-Hardness of Edge-MstP with Vertex Cover Number, k, and ℓ as Parameters

Let ν be the vertex cover number of the underlying graph $G_\downarrow$ of the temporal graph G in an input instance of Edge-MstP. Then Edge-MstP is W[1]-hard when parameterised by $\nu + k + \ell$.

We begin by describing the construction which will be used as polynomial parameter transformation:

Construction 3.17. Let $(G = (V, E), k, c)$ be an instance of Multicoloured Clique. Let $n = |V|$. Let $k' = 6k - 2$ and $\ell = 3k + 2$. The parameters k' and ℓ will be used in the output instance, note that both are linear in k. Let $H = [3k - 5]$, $A = \bigcup_{d \in [k]} a_d$ and $B = \bigcup_{d \in [k]} b_d$. We then construct a temporal graph G' with vertex set $V' = V \cup \{s, t\} \cup A \cup B \cup H$ and edge sets $E_1, \dots, E_{2n(k-1)}$.

Each odd layer G_i shall ensure that there is exactly one vertex of each colour in any st-path in G_i. Also each $v \in V$ with colour $c(v) = j$ shall only be adjacent to a_j and b_j. Moreover, all odd layers of G' are identical, i.e. $E_1 = E_3 = \cdots = E_{2n(k-1)-1}$. To this end, let

$$E' := \{\{s, 1\}, \{t, k + 1\}\} \cup \bigcup_{d \in [k]} \bigcup_{\substack{w \in V \\ c(w) = d}} \{\{d, a_d\}, \{a_d, w\}, \{w, b_d\}, \{b_d, d + 1\}\}.$$

and let $E_i = E'$ for each odd $i \in [2n(k - 1)]$.

Next there shall be one even layer for each $v \in V$ and $j \in [k] \setminus \{c(v)\}$, totalling to $n(k - 1)$ even layers in G'. Thus, we construct an edge set E^v_j for each $v \in V$ and $j \in [k] \setminus \{c(v)\}$. Later, if v is included in an st-path P in the resulting layer, then P shall also include a vertex $u \in V$ with colour j so that $\{u, v\} \in E$. The vertices in H are used in order to make the shortest st-path in the resulting layer contain exactly ℓ vertices.

$$E^v_j = \{\{s, b_{c(v)}\}, \{b_{c(v)}, v\}, \{v, a_j\}, \{a_{c(v),1}, \{3k - 5, t\}\}\} \cup \bigcup_{\substack{w \in V \setminus \{v\} \\ c(w) = c(v)}} \{\{b_{c(v)}, w\}, \{w, b_j\}\} \cup$$
$$\bigcup_{\substack{w \in V \\ c(w) = j}} \{\{b_j, w\}, \{w, a_{c(v)}\}\} \cup \bigcup_{\substack{w \in V \\ c(w) = j \\ \{v,w\} \in E}} \{\{a_j, w\}\} \cup \bigcup_{h \in [3k-4]} \{h, h + 1\}$$

We use each E^v_j as the edge set for a different even layer in G' (their order is irrelevant).

This finishes the construction, the output instance is (G', s, t, k', ℓ). An example of this construction is illustrated in Figure 3.3. ♦

The following proofs are mostly similar to the ones in the previous subsection: We will prove the claim that Construction 3.17 is a polynomial parameter transformation from

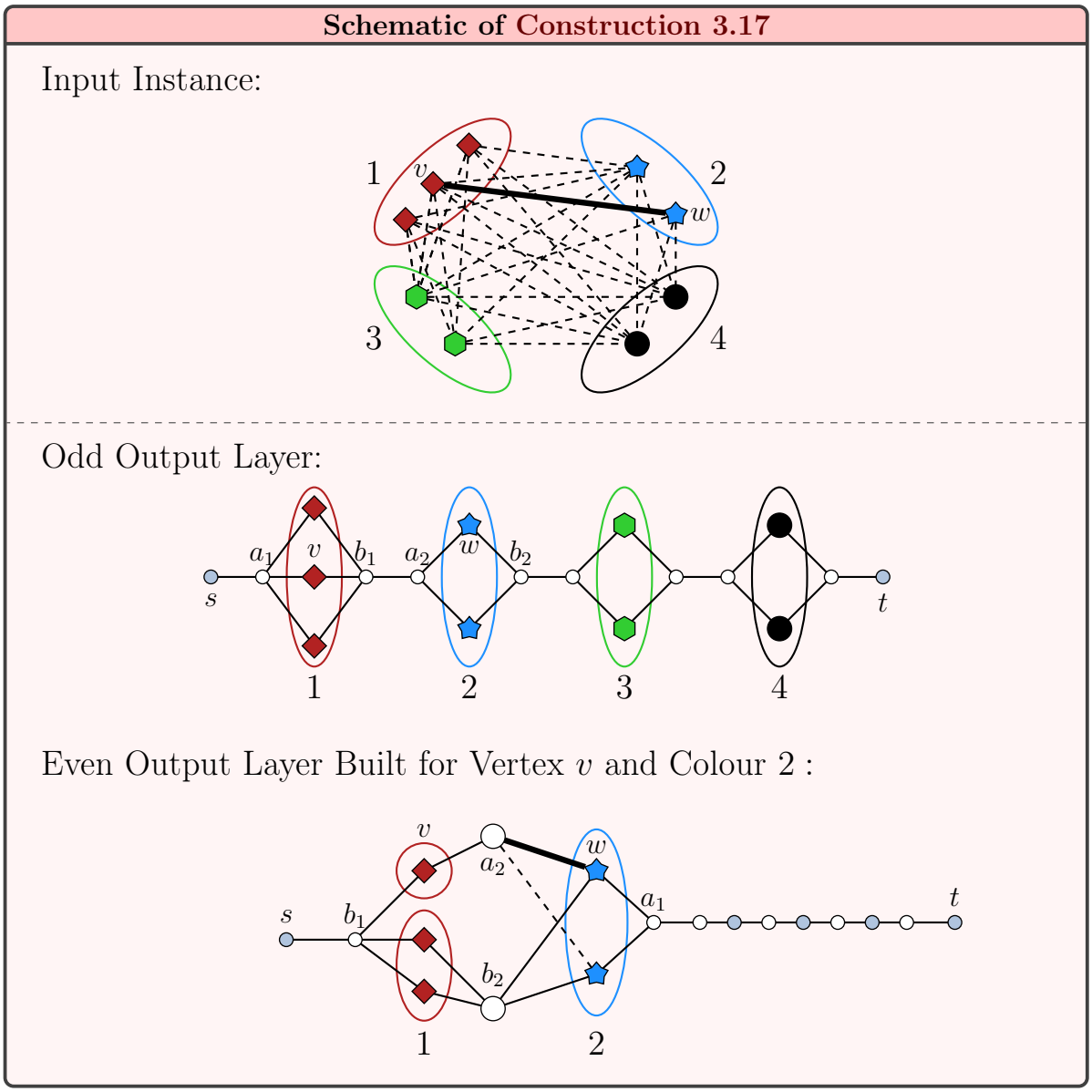


Figure 3.3: In this picture, examples of an odd and an even output layer for a small input instance are illustrated. The vertices in the minimum vertex cover are coloured white, whereas vertices in V are dyed and shaped according to their respective colour. The edges indicated by dashed lines in the input may exist (but do not have to), whereas the thick line represents an edge that does exist. The illustrated even output layer is built specifically for vertex v with $c(v) = 1$ and colour $c(w) = 2$. It contains the edge $\{a_2, w\}$, because in the input the edge $\{v, w\}$ exists (indicated by a thick line). It may also contain an edge $\{a_2, u\}$ for any $u \in V$ if $c(u) = 2$ and an edge $\{v, u\}$ is in the input graph (indicated by a dashed line). Vertices which are of degree zero in a layer are not shown in that layer, e.g. the vertices in H which form a path between a_2 and t in the depicted even layer are of degree zero and thus not shown in the depicted odd layer.

Multicoloured Clique parameterised by k to Edge-MstP parameterised by $\nu+k'+\ell$ (as defined in the construction) and for this reason show that all paths within a solution to Edge-MstP for the output instance include the same vertices from the input vertex set V. However, this time we also use to vertices in H together with the length parameter ℓ of Edge-MstP to guarantee this:

Observation 3.18. If P is an st-path with $|V(P)| \leq \ell$ in an even layer G_i built by Construction 3.17 specifically for some vertex v and colour j, then P includes exactly one vertex with colour $c(v)$ and exactly one vertex with colour j.

Proof. Let $X = \{x \in V : c(x) = c(v)\}$ and $Y = \{y \in V : c(y) = j\}$. As X and Y are both st-separators in G_i, at least one $x \in X$ and one $y \in Y$ have to be included in P. We also see that P includes at least a_j or b_j and furthermore, $\{s, t, b_{c(v)}, a_{c(v)}\} \cup H \subseteq V(P)$. This means that $V(P) \setminus (X \cup Y)$ already includes $3k$ vertices. As $V(P) \leq \ell = 3k + 2$, P can consequently only include one $x \in X$ and one $y \in Y$. □

As already indicated, we can now prove the most important lemma for the subsequent correctness proof of Construction 3.17, which states that within a solution of Edge-MstP for the output instance all paths include the same vertices from V:

Lemma 3.19. If $S = (P_1, \ldots, P_\tau)$ is a solution to Edge-MstP for an output instance $(G' = (V', E_1, \ldots, E_\tau), s, t, k', \ell)$ of Construction 3.17, then it holds that $V(P_1) \cap V = \cdots = V(P_\tau) \cap V$.

Proof. Let $i \in [\tau - 1]$. Let G_i and G_{i+1} be the i-th and $(i+1)$-th layer in the constructed temporal graph G'. Clearly, one of them is even and the other is odd. Without loss of generality, assume that G_i is the odd layer. The corresponding st-path P_i in S clearly includes exactly one vertex of each colour and $3k + 1$ edges in total.

Since G_{i+1} is an even layer, it was constructed for a particular vertex v and colour j. By Observation 3.18 the corresponding st-path P_{i+1} with $|V(P_{i+1})| \leq \ell$ in S includes exactly one $x \in V$ with colour $c(v)$ and one $y \in V$ with colour j. We next distinguish between whether $x = v$ or $x \neq v$:

Case $x = v$: Then P_{i+1} goes from s via $b_{c(v)}$, v, a_j, y, $a_{c(v)}$, and the vertices in H to t (in that order). Then the edges which are both included in P_i and in P_{i+1} can only be $\{b_c(v), v\}$ and $\{a_j, y\}$, as each other edge is either not included in P_{i+1} or not present in G_i. Since P_i and P_{i+1} include $3k + 1$ edges each and only two of them can be in both, it holds that

$$\begin{aligned} |E(P_i) \triangle E(P_{i+1})| &= |(E(P_i) \setminus E(P_{i+1})) \cup (E(P_{i+1}) \setminus E(P_i))| \\ &\geq (3k + 1 - 2) + (3k + 1 - 2) = 6k - 2 = k'. \end{aligned}$$

Since the edge difference of P_i and P_{i+1} is at most k', the edges $\{b_c(v), v\}$ and $\{a_j, y\}$

indeed have to be included in P_i as well as in P_{i+1}. Thus we conclude that v and y are both included in P_i and P_{i+1}, which proves the statement above for this case.

Case $x \neq v$: There is no difference to the previous case, except that P_{i+1} now goes from s via $b_{c(v)}$, x, b_j, y, $a_{c(v)}$, and the vertices in H to t (in that order) and the two edges which have to be included in both P_i and P_{i+1} are now $\{b_c(v), x\}$ and $\{b_j, y\}$, respectively. □

We now show that Construction 3.17 is correct:

Lemma 3.20. The output of Construction 3.17 is a YES-instance of EDGE-MSTP if and only if its input is a YES-instance of MULTICOLOURED CLIQUE.

Proof. ($\Leftarrow$) If there is a multicoloured clique C of size k in the input graph $G = (V, E)$, then let $R = C \cup A \cup B \cup \{s, t\}$. In each constructed odd layer G_i, R clearly forms an st-path P_i with $|V(P_i)| = 3k + 2 = \ell$ (as R contains exactly one vertex of each colour) and consequently $|E(P_i)| = 3k + 1$.

Now consider G_{i+1}, which is an even layer specifically constructed for some vertex $v \in V$ and some colour $j \in [k] \setminus \{c(v)\}$. Let $x \in C$ be the vertex in C which has colour $c(v)$ and let $y \in C$ be the vertex in C which has colour j. If $x = v$, there is an st-path P_{i+1} with $|E(P_{i+1})| = 3k + 1$ going from s via $b_{c(v)}$, v, a_j, y, $a_{c(v)}$, and the vertices in H to t (in that order) in G_{i+1}. Note that the edge $\{a_j, y\}$ in G_{i+1} exists, because an edge $\{x, y\}$ exists in the input graph. Then $E(P_i) \cap E(P_{i+1}) = \big\{\{b_{c(v)}, v\}, \{a_j, y\}\big\}$ and the edge difference between P_i and P_{i+1} is $(3k - 1) + (3k - 1) = 6k - 2 \leq k'$.

Otherwise ($x \neq v$), there is an st-path P'_{i+1} with $|E(P'_{i+1})| = 3k + 1$ in G_{i+1} instead, which goes from s via $b_{c(v)}$, x, b_j, y, $a_{c(v)}$, and the vertices in H to t (in that order). Then $E(P_i) \cap E(P'_{i+1}) = \big\{\{b_{c(v)}, x\}, \{b_j, y\}\big\}$ and the edge difference between P_i and P'_{i+1} is again $(3k - 1) + (3k - 1) = 6k - 2 \leq k'$.

We thus find a solution to EDGE-MSTP for the constructed instance this way, which notably contains the same st-path in each odd layer, but different st-paths in the even layers.

($\Rightarrow$) If the input is a NO-instance of MULTICOLOURED CLIQUE, assume towards a contradiction that an output was a YES-instance of EDGE-MSTP. Then there is a corresponding solution $S = (P_1, \ldots, P_\tau)$ for that output instance.

Let $C = V(P_1) \cap V$. We know that C consists of exactly one vertex of each colour, otherwise P_1 could be no st-path in the first constructed layer. However, C forms no MULTICOLOURED CLIQUE in the input graph $G = (V, E)$ (as the input is a NO-instance). Then there are two distinct vertices $a, b \in C$ so that $\{a, b\} \notin E$.

Let G_i be the even layer constructed specifically for vertex a and colour $c(b)$. Since S is a solution, there is an st-path P_i with $|V(P_i)| \leq \ell$ in G_i so that by Lemma 3.19 P_i also includes both a and b. Moreover, P_i does not even include any other vertices of colours $c(a)$ and $c(b)$ (Observation 3.18). However, since $\{a, b\} \notin E$ there is no constructed edge $\{a_{c(b)}, b\}$ in G_i. As the layer was constructed for vertex $v = a$, there is no edge $\{a, b_{c(b)}\}$ either. With this we see that P_i can neither proceed via $a_{c(b)}$ nor

via $b_{c(b)}$ and thus not includes both a and b without including other vertices of colours $c(a)$ and $c(b)$, which is a contradiction. □

Having dealt with the issue of correctness, we of course also note that this construction can be executed in polynomial time:

Observation 3.21. Construction 3.17 can be applied in $O(n^3)$ time, where n is the number of vertices in the input instance.

Proof. The number of constructed vertices is at most $6n - 3$, since the number of colours is at most n. In each constructed odd layer each $v \in V$ is of degree two, so there are $2n$ edges with endpoints in V. Additionally, there are at most $2n+1$ edges between vertices in $A \cup B \cup \{s, t\}$. Therefore constructing each odd layer takes time in $O(n)$.

In each constructed even layer each $v \in V$ is at most of degree two, so there are at most $3n$ edges with endpoints in V. Additionally, there are at most $3n - 3$ edges with endpoints in $\{s\} \cup H$. Thus constructing each even layer also takes time in $O(n)$. Since there are at most $2n(n-1) \in O(n^2)$ layers in the output, the total running time is in $O(n^3)$. □

Now we are set for the proof for our claim, i.e. for Construction 3.17 being an appropriate polynomial parameter transformation:

Lemma 3.22. Construction 3.17 is a polynomial parameter transformation from Multicoloured Clique parameterised by k to Edge-MstP parameterised by $\nu + k' + \ell$, where k is the number of colours in Multicoloured Clique, k' is the edge difference parameter of Edge-MstP, ℓ is the path length parameter of Edge-MstP, and ν is the vertex cover number of the underlying graph of the temporal graph in instances of Edge-MstP.

Proof. Let $G_\downarrow$ be the underlying graph of a temporal graph G built by Construction 3.17 for an input instance $(G = (V, E), k, c)$. We denote the vertex sets H, A, and B as in the construction. We construct a vertex cover of size $2k + \left\lceil \frac{3k-5}{2} \right\rceil$ in $G_\downarrow$ as follows:

If k is an odd number, let $D := \{h \in H : \text{ h is an odd number}\}$. Otherwise, let $D := \{h \in H : \text{ h is an even number}\}$. Let

$$C := D \cup A \cup B.$$

Then C forms a vertex cover in $G_\downarrow$, as each edge in each layer G_i of G has an endpoint in C.

Since C is a vertex cover of size $2k + \left\lceil \frac{3k-5}{2} \right\rceil$ in $G_\downarrow$ and the parameters k' and ℓ of Edge-MstP are assigned to $6k - 2$ and $3k + 2$ in Construction 3.17, it holds that $\nu + k' + \ell \leq 11k + \left\lceil \frac{3k-5}{2} \right\rceil \leq 12.5k$, so $\nu + k' + \ell$ is polynomially upper-bounded in k. Together with Lemma 3.20 and Observation 3.21 this means that Construction 3.17 is

a polynomial parameter transformation from Multicoloured Clique parameterised by k to Edge-MstP parameterised by $\nu + k' + \ell$. □

The main theorem of this subsection, i.e. Theorem 3.16, is now a direct result:

Proof of Theorem 3.16. By Lemma 3.22 we know that there is a polynomial parameter transformation from the W[1]-hard Multicoloured Clique parameterised by the number of colours to Edge-MstP parameterised by $\nu + k + \ell$. Thus, Edge-MstP parameterised by $\nu + k + \ell$ is also W[1]-hard. □

In contrast to Construction 3.11, Construction 3.17 does not assign the parameter k' of its output instance to a constant. Hence, it remains open if Edge-MstP is also W[1]-hard when parameterised by $\nu + \ell$ and k is some constant $c \in \mathbb{N}$. We also did not study Leven-MstP with respect to $\nu + k + \ell$ in this section. Presumably, another construction similar to Construction 3.17 can be described in order to show W[1]-hardness of Leven-MstP for the parameter $\nu + k + \ell$, but this remains an open question for now.

4 Efficient Preprocessing and Fixed-Parameter Tractability

In this chapter, we present certain kernelisation results for Multistage st-Path. In particular, we show in Section 4.1 that our variants of Multistage st-Path each admit an exponential-sized problem kernel when parameterised by the number of input layers and the vertex cover number of the underlying graph of the input temporal graph. Subsequently, we prove in Section 4.2 that no polynomial-sized kernel exists for Vertex-MstP with those parameters combined (unless the polynomial hierarchy collapses). In Section 4.3 we provide an FPT-algorithm for Multistage st-Path with respect to the maximal degree over all input layers combined with the path length. Finally, in Section 4.4 we present a problem kernel of Vertex-MstP with size polynomial in the feedback edge number of the underlying graph of the input temporal graph combined with the number of input layers.

4.1 Exponential Kernel for Multistage st-Path in the Vertex Cover Number and Number of Layers

We showed that Multistage st-Path (with both vertex and edge distance) is W[1]-hard when parameterised either by the vertex cover number ν of the underlying graph $G_{\downarrow}$ (Theorems 3.10 and 3.16) or by the number of layers τ of G (Theorem 3.1 and Cor. 3.8).

Hence, we wonder whether there is an FPT-algorithm for Vertex-MstP when it is parameterised by ν and τ combined. In this case, we can answer in the affirmative. More specifically, we will show that there is a kernelisation of Vertex-MstP when parameterised by these two parameters:

Theorem 4.1: Exponential-sized Kernel for Vertex-MstP parameterised by Vertex Cover Number and Number of Layers

Vertex-MstP admits a problem kernel with at most $|V| \leq (2^{\tau\nu}+1)\nu+2$ vertices, where ν is the vertex cover number of the underlying graph of the input temporal graph and τ is the number of layers in the input.

This section is thus dedicated to proving Theorem 4.1. But in order to do so, we first need some preprocessing involving *temporal twin vertices*. This concept will be introduced in the following Subsection 4.1.1. In Subsection 4.1.2, we will present the polynomial-time Algorithm 3 which computes these temporal twin vertices. We later use these vertices

in Subsection 4.1.3 for finding an appropriate problem kernel of Vertex-MstP, which then proves Theorem 4.1. We also find similar problem kernels for Edge-MstP and Leven-MstP (Corollary 4.7).

4.1.1 Temporal Twins

In classic graph theory two vertices are often called *twins* if their neighbourhoods coincide. More precisely, one often distinguishes *adjacent twins* and *non-adjacent twins* in simple graphs (see e.g. Hammer and Maffray (1990) and Hernando et al. (2007)):

Definition 4.2: Twin Vertices

Two distinct vertices u, v of a graph $G = (V, E)$ are called *adjacent twins* if $N[u] = N[v]$. They are called *non-adjacent twins* if $N(u) = N(v)$.

Note that those names are justified, because two adjacent twins always share an edge in G, whereas non-adjacent twins never do (by Definition 4.2).

We now introduce a notion of twins to temporal graph theory. To the best of our knowledge, this is the first time such a concept is utilised on temporal graphs. In particular, we want to call two vertices u, v in a temporal graph G *temporal twins* if within each layer of G the neighbourhoods of u and v coincide.

Definition 4.3: Temporal Twin Vertices

Two distinct vertices u, v of a temporal graph $G = (V, E_1, \ldots, E_\tau)$ are called *adjacent temporal twins* if for all $i \in [\tau]$ it holds that $N_{G_i}[u] = N_{G_i}[v]$. They are called *(non-adjacent) temporal twins* if for all $i \in [\tau]$ it holds that $N_{G_i}(u) = N_{G_i}(v)$.

In the following subsections, we will only consider non-adjacent temporal twins. We will next present a simple polynomial time algorithm for finding temporal twins in any temporal graph and later use them for kernelisation of the Vertex-MstP problem.

4.1.2 Algorithm for Finding Temporal Twins

Algorithm 3 consists of two procedures. The first one, GetTwins, finds all twins of a vertex v in a simple graph. The second procedure, GetTemporalTwins, calls GetTwins separately on each layer in the input temporal graph. It then returns all vertices that were in each result of those sub-procedure calls (and thus are a twin of v in each layer).

This way, we find all temporal twins of v in G.

Algorithm 3: The procedure GetTemporalTwins finds all temporal twins of a vertex v in a temporal graph G.

```
1  Procedure GetTwins(G', v)
       Input  : A simple graph G' = (V, E') and a vertex v ∈ V.
       Output : A set S' containing all twin vertices of v in G'.
2      S' ← {}
3      forall w ∈ V \ {v} do
4          if N(v) = N(w) then                    /* comparison in O(N(w)) time */
5              add w to S'
6          end
7      end
8      return S'
9  Procedure GetTemporalTwins(G, v)
       Input  : A temporal graph G = (V, E_1, ..., E_τ) and a vertex v ∈ V.
       Output : A set of all temporal twins of v in G.
10     S ← GetTwins((V, E_1), v)                  /* takes O(|E_1|) time */
11     for i ← 2 to τ do
12         S ← S ∩ (GetTwins((V, E_i), v))        /* takes O(|E_i| + |S|) time */
13     end
14     return S
```

Lemma 4.4. The procedure GetTemporalTwins in Algorithm 3 finds all non-adjacent temporal twins of a vertex v in a temporal graph G in $O(m)$ time (assuming all input sets are ordered), where m is the total number of temporal edges in G, i.e. $m = \sum_{i \in [\tau]} |E_i|$.

Proof. We split this proof into two parts, one for showing the correctness of Algorithm 3 and one for showing its above stated running time.

Correctness: We prove that the output set S of procedure GetTemporalTwins in Algorithm 3 equals the set of non-adjacent temporal twins of v in G:

($\Rightarrow$) Let $w \in S$. We know that S is the intersection of all sets computed by the sub-procedure GetTwins, which was called once for each (G_i, v), $i \in \tau$ (lines 10 to 13 in Algorithm 3). Let G_i be any layer of G and let S' be the corresponding output set of GetTwins for (G_i, v). Since $S \subseteq S'$, it follows directly that $w \in S'$. Then by specification of GetTwins (lines 3 to 7 in Algorithm 3) it holds that $w \in V \setminus \{v\}$ and that $N_{G_i}(v) = N_{G_i}(v)$. Thus, w is a temporal twin of v in G by Definition 4.3.

($\Leftarrow$) Let w be a non-adjacent temporal twin of v in G. Let G_i be any layer of G. Then w is a non-adjacent twin of v in G_i. Thus, the sub-procedure GetTwins in Algorithm 3 outputs a set S' with $w \in S'$ when it is called for the input (G_i, v) (lines 3 to 7 in

Algorithm 3). Since G_i was chosen as an arbitrary layer in G, w is in the output set of each GetTwins call done by the GetTemporalTwins procedure. Then w is also in their set intersection and consequently in the output set S of GetTemporalTwins (lines 10 to 13 in Algorithm 3).

Running time: We assume that V and all sets $N(v)$ for any $v \in V$ are already ordered in the input. Thus, their set comparisons and set intersections can be computed in linear time. The sub-procedure GetTwins takes $O(|E'|)$ time on an input graph $G' = (V, E')$, as comparing two ordered sets $N(v)$ and $N(w)$ can be done in $O(N(w))$ time and is done once for each $w \in V \setminus \{v\}$.

The main procedure GetTemporalTwins hence takes time in $O(|E_1|)$ on the first layer and time in $O(|E_i| + |E_{i-1}|)$ on each subsequent layer $G_i = (V, E_i)$, $i \in [2, \tau]$ (by calling the sub-procedure and then intersecting). Thus the total running time is $O(\sum_{i=2}^{\tau}(|E_i| + |E_{i-1}|) + |E_1|) = O(\sum_{i=1}^{\tau} |E_i|)$ which proves Lemma 4.4. □

4.1.3 Kernelisation

We now use temporal twins to formulate a new data reduction rule for Vertex-MstP. Intuitively, if two vertices are temporal twins, then they can be exchanged for each other in the paths of any given solution. This can be used for preprocessing, because each path can contain only a limited number of such vertices (trivially ℓ, but we will narrow this down more later). Therefore we can delete vertices if they can be replaced by their temporal twins in solution paths.

Data Reduction 4.5 (Twin Reduction): Let $S[v]$ be the set of all non-adjacent temporal twins of some vertex $v \in V \setminus \{s, t\}$ together with v itself. Let $d(v)$ be the maximal degree of v over all layers, i.e. $d(v) = \max_{i \in [\tau]} \deg_{G_i}(v)$. If $d(v) \leq |S[v] \setminus \{s, t\}|$, then delete v from G.

Lemma 4.6. Data Reduction 4.5 is *safe* and can be applied once on each $v \in V \setminus \{s, t\}$ in $O(nm)$ time (assuming all input sets are ordered), where $n = |V|$ and $m = \sum_{i \in [\tau]} |E_i|$.

Proof. Let $v, S[v], d(v)$ be as stated in Data Reduction 4.5. Let $S = S[v] \setminus \{s, t\}$.

Now let P be an st-path in any layer. For every vertex $u \in V(P) \cap S$ there also exists a distinct vertex $w \in V(P) \cap N(v)$, because all vertices in S are pairwise non-adjacent. Obviously, P begins and ends outside S (as $s, t \notin S$). This means that P contains more vertices from $N(v)$ than from S. We assume that $d(v) \leq |S|$ (since Data Reduction 4.5 does nothing otherwise). Then P consequently does not include each vertex in S.

Now let M be a "vertex-minimal" solution for the instance, that is, among all solutions M minimises the total number of vertices included over its st-paths. Then we especially note that there is a vertex $u \in S$ which is not included in any st-path

in M, since all vertices in S are temporal twins and thus exchangeable for each other. If $u = v$, then v can be safely deleted (there is still a solution to the reduced instance if and only if M exists). Otherwise, v can be exchanged for u in each st-path in M that included v, without changing any path distance value. This way, we find another solution in which v is not used (if and only if M exists). Hence, v can be safely deleted in this case as well.

The stated running time follows from Lemma 4.4 if the procedure GetTemporalTwins is called once on each $v \in V \setminus \{s, t\}$. □

Applying Data Reduction 4.5 on each $v \in V \setminus s, t$ is sufficient to obtain a problem kernel with $|V| \leq (2^{\tau\nu} + 1)\nu + 2$. Hence, we can now prove Theorem 4.1:

Proof of Theorem 4.1. We apply Data Reduction 4.5 exhaustively beforehand, which takes polynomial time as stated in Lemma 4.6 (if any sets are not necessarily ordered, then we sort them prior to the application of Data Reduction 4.5).

Let $C \subseteq V$ be a vertex cover for $G_{\downarrow}$ of minimal size. Then $|C| = \nu$ and the maximal degree in $G_{\downarrow}$ (and in any layer) is also at most ν. Now let $I = V \setminus C$. We note that no two distinct vertices in I are adjacent. We now "distribute" the vertices in $I \setminus \{s, t\}$ into categories according to exactly which neighbours they have in exactly which layers.

There are at most $(2^{\nu})^{\tau} = 2^{\tau\nu}$ such categories, because every combination of any number of available layers and vertices in C may form a category. Now all vertices within the same category are non-adjacent temporal twins, because they have the same neighbours in each layer.

Due to the previous application of Data Reduction 4.5, we then know that in each category there are at most ν vertices left (since $N(v) \subseteq C$ for all $v \in I$). Thus, it holds that $|I \setminus \{s, t\}| \leq 2^{\tau\nu}\nu$. As $|C| \leq \nu$ we then infer that $|V| = |I| + |C| \leq (2^{\tau\nu} + 1)\nu + 2$, which proves Theorem 4.1. □

The safeness of Data Reduction 4.5 (see proof of Lemma 4.6) does not rely on the particular distance function of Vertex-MstP, that is, the vertex distance. The only "requirement" on the distance function in order to allow for application of Data Reduction 4.5 is that temporal twins are "treated equally", meaning each vertex u can be safely exchanged with its temporal twin w in all paths in any given solution without altering any distance function value. As a side note, Data Reduction 4.5 would e.g. not be safe in a hypothetical weighted version of Multistage st-Path where the distance function depends on different vertex or edge weights. Since this is clearly not the case for Edge-MstP and Leven-MstP, Data Reduction 4.5 can also be safely applied on instances of these two problems.

Thus, the proof of Theorem 4.1 also proves the following:

Corollary 4.7: Exponential-sized Kernels for Edge-MstP and Leven-MstP parameterised by Vertex Cover Number and Number of Layers

Edge-MstP and Leven-MstP both admit problem kernels with each at most $|V| \leq (2^{\tau\nu} + 1)\nu + 2$ vertices, where ν is the vertex cover number of the underlying graph of the input temporal graph and τ is the number of layers in the input.

It is known that a parameterised problem has a kernelisation if and only if it is fixed-parameter tractable and decidable (see e.g. the survey by Kratsch (2014)). Thus, from Theorem 4.1 and Corollary 4.7 we also infer that Vertex-MstP, Edge-MstP and Leven-MstP are fixed-parameter tractable when parameterised by $\nu + \tau$.

4.2 No Kernel of Size Polynomial in Vertex Cover Number and Number of Layers for Vertex-MstP

In the previous section we showed that Vertex-MstP admits a problem kernel when parameterised by the vertex cover number of the underlying graph of the input temporal graph combined with the number of layers of the input temporal graph, i.e. $\nu + \tau$ (Theorem 4.1). The kernel is of size exponential in $\nu + \tau$, however. Next we show that a polynomial kernel appears unlikely, as its existence would imply a collapse of the polynomial hierarchy:

Theorem 4.8: No polynomial-sized Kernel for Vertex-MstP parameterised by Vertex Cover Number and Number of Layers

Vertex-MstP does not admit a problem kernel of size polynomial in $\nu+\tau+k+\ell$, unless coNP $\subseteq$ NP/poly. Herein, ν is the vertex cover number of the underlying graph of the input temporal graph, τ is the number of layers in the input, k is the maximal vertex distance between adjacent paths in a solution and ℓ is the maximal length of paths in a solution.

We prove Theorem 4.8 using a *polynomial equivalence relation* and an or-*cross-composition* as defined by Bodlaender, Jansen, and Kratsch (2014):

Definition 4.9

Let Σ be a finite alphabet. An equivalence relation $\mathcal{R}$ on Σ^* is called a polynomial equivalence relation if the following two conditions hold:

- There is an algorithm that given two strings $x, y \in \Sigma^*$ decides whether x and y belong to the same equivalence class in time polynomial in $|x| + |y|$.
- For any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of S into a number of classes that is polynomially bounded in the size of the largest element of S.

Definition 4.10

Let Σ be a finite alphabet. Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on Σ^*, and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterised problem. An or-cross-composition of L into Q (with respect to R) is an algorithm that, given t input instances $x_1, \ldots, x_t \in \Sigma^*$ of L belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that the following conditions hold:

- The parameter value k is polynomially upper-bounded in $\max_{i=1}^{t} |x_i| + \log t$.
- (y, k) is a yes-instance for Q if and only if at least one input instance x_i is a yes-instance for L.

Our results build upon the following theorem proven by Bodlaender, Jansen, and Kratsch (2014):

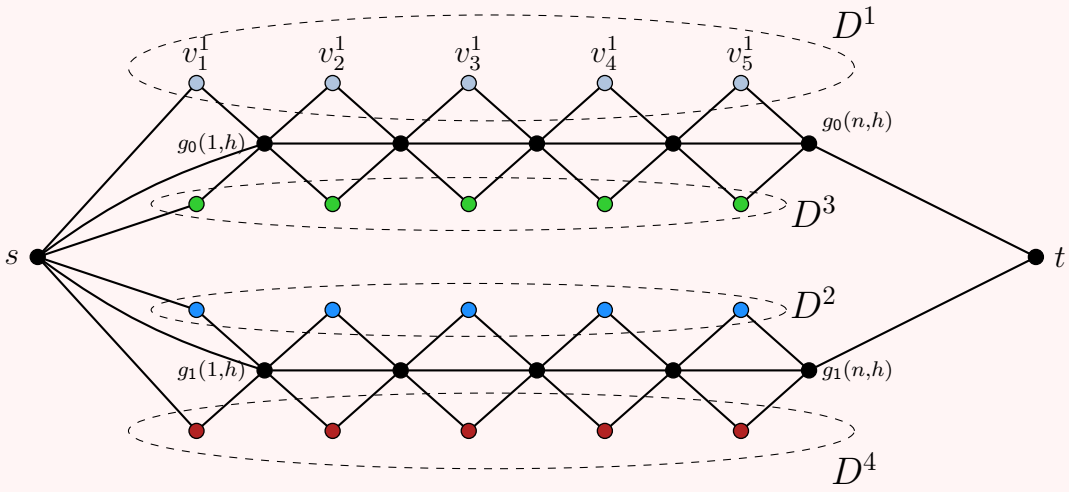
Theorem 4.11

If an NP-hard language L has an or-cross-composition into the parameterised problem Q, then Q does not admit a polynomial kernelisation or polynomial compression unless coNP $\subseteq$ NP/poly.

Next we describe an or-cross-composition into Vertex-MstP parameterised by $\nu + \tau + k + \ell$. We choose Positive 1-in-3-SAT (from Definition 2.14) as the NP-hard language L for Theorem 4.11. Our polynomial equivalence relation $\mathcal{R}$ then contains (x, y) if the two strings $x, y \in \Sigma^*$ encode instances of Positive 1-in-3-SAT that have the same number of variables and clauses each. Deciding if $(x, y) \in \mathcal{R}$ takes polynomial time, as it is only necessary to check whether the numbers of variables and clauses in the instance encoded by x equals the numbers of variables and clauses in the instance encoded by y. There are $nm + 1$ equivalence classes (where n and m are the respective numbers of variables and clauses of the largest instance x_i, the remaining class shall be a garbage class containing all ill-formated inputs). Therefore $\mathcal{R}$ is a valid polynomial equivalence relation. Having established these requirements, we can now start the construction:

Schematic of Construction 4.12

(a) Example layer for $G_1, \ldots, G_{(\log p)}$:



(b) Example layer for $G_{(\log p)+1}, \ldots, G_{(\log p+m)}$:

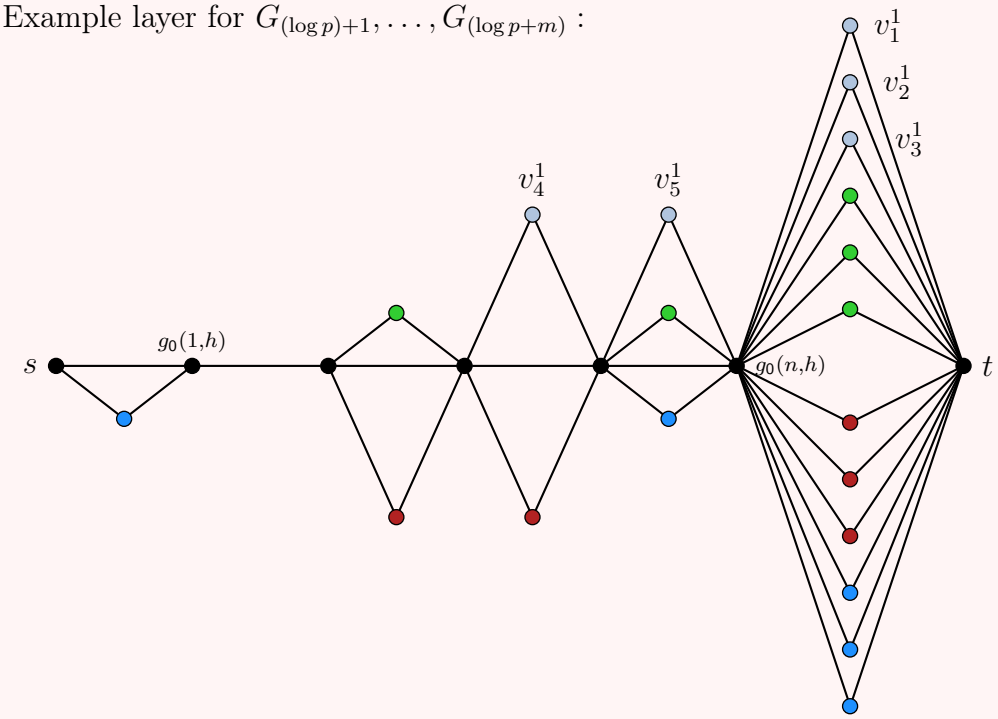


Figure 4.1: Two example layers generated by Construction 4.12 are depicted. Isolated vertices are not shown. Vertices in the minimal vertex cover $V \setminus D$ are illustrated as black, whereas the vertices in D are coloured according to which input instance they correspond to. The example layer in (b) is generated for some clause $C_j^1 = \{x_1, x_2, x_3\}$ in the corresponding input instance a_1.

Construction 4.12. An example for this construction is illustrated in Figure 4.1.

Let $a_1, \ldots, a_p$ be encoded instances of Positive 1-in-3-SAT, each of them having m clauses and n variables. We assume p to be a power of two. Otherwise it would be possible to "repeat" the first instance until p is a power of two, which at most doubles the number of instances (and therefore effectively does not influence the construction complexity later). We refer to the clause C_j in instance a_q as C_j^q and to the variable x_i in instance a_q as x_i^q.

We need a vertex for each variable in each instance, similar to Construction 2.16. These vertices will not be part of a minimum vertex cover and hence their number does not need to be polynomially upper-bounded in $\max_{q=1}^{p} |a_q| + \log p$. Let

$$D^q := \{v_i^q : i \in [n]\} \text{ for all } q \in [p] \text{ and}$$
$$D := \bigcup_{q=1}^{p} D^q.$$

Each D^q represents all variables of the input instance a_q and therefore D represents all variables over the p input instances. Including some vertex $v_i^q \in D$ to an st-path within a solution shall later represent the corresponding variable x_i^q to be assigned to *true*.

As those vertices in D shall not be part of our minimum vertex cover (but part of st-paths), there shall be no two adjacent vertices $x, y \in D$ in any constructed layer. Instead we need other special vertices that are connected to the vertices in D. As those will be part of the minimum vertex cover, their number must be polynomially upper-bounded in $\max_{q=1}^{p} |a_q| + \log p$. To this end, we build the vertex sets E' and F', which contain $2n$ unique vertices each. Let

$$E' := \bigcup_{i=1}^{n} \{e_0^i, e_1^i\}, F' := \bigcup_{i=1}^{n} \{f_0^i, f_1^i\} \text{ and}$$
$$V = E' \cup F' \cup D \cup \{s, t\}.$$

The set V is the vertex set of the constructed instance.

The vertices in E' are used in even layers and the vertices in F' are used in odd layers. This means that in each layer only vertices of one of these two sets have edges, depending on if that layer is odd or even. To simplify this procedure, we define the helper functions $g_0, g_1 : [n] \times \mathbb{N} \to E' \cup F'$ where

$$g_b(i, h) := \begin{cases} e_b^i & \text{if } h \text{ is even,} \\ f_b^i & \text{if } h \text{ is odd,} \end{cases} \quad \text{where } b \in \{0, 1\}.$$

We now begin constructing the edge sets $E_1, \ldots, E_{\log p + m}$ (which will be used in the output instance).

First let $h \in [\log p]$. The edge set E_h shall contain two paths from s to t, one via

$g_0(1,h),\ldots,g_0(n,h)$ and one via $g_1(1,h),\ldots,g_1(n,h)$:

$$\text{LINES}_h := \{\{s,g_0(1,h)\},\{s,g_1(1,h)\},\{t,g_0(n,h)\},\{t,g_1(n,h)\}\} \cup \bigcup_{i\in[n-1]} \{\{g_0(i,h),g_0(i+1,h)\},\{g_1(i,h),g_1(i+1,h)\}\}$$

In order to make these paths extendable by the vertices in D we subdivide D into exactly two disjoint subsets $S_0^h, S_1^h \subseteq D$. Let $S_0^h = \{v_i^q \in D : q-1$ has a 0 at the h-th position if $q-1$ is binarly encoded$\}$. Let $S_1^h = D \setminus S_0^h$, which means that S_1^h is the set containing every vertex $v_i^q \in D$ where $q-1$ has a 1 at the h-th position if $q-1$ is binarly encoded.

Let $X_i := \bigcup_{q\in[p]} \{v_i^q \in D\}$ for each $i \in [n]$. That means each X_i contains each vertex corresponding to any variable $x_i^q,\ q \in [p]$.

With that, we build the two remaining edge sets required for E_h. Basically, we make each vertex $u \in D$ adjacent to two vertices in $E' \cup F'$, depending on whether $u \in S_0^h$ or $u \in S_1^h$. If $u \in S_0^h$, u gets two neighbours with the index 0 (in edge set UP_h). Otherwise, the neighbours index is 1 (in edge set LOW_h).

$$\text{UP}_h := \bigcup_{x\in X_1\cap S_0} \{\{s,x\},\{x,g_0(1,h)\}\} \cup \bigcup_{i\in[n-1]} \bigcup_{x\in X_{i+1}\cap S_0} \{\{g_0(i,h),x\},\{g_0(i+1,h),x\}\}$$

$$\text{LOW}_h := \bigcup_{x\in X_1\cap S_1} \{\{s,x\},\{x,g_1(1,h)\}\} \cup \bigcup_{i\in[n-1]} \bigcup_{x\in X_{i+1}\cap S_1} \{\{g_1(i,h),x\},\{g_1(i+1,h),x\}\}$$

By combining these parts we get E_h for all $h \in [\log p]$:

$$E_h = \text{LINES}_h \cup \text{UP}_h \cup \text{LOW}_h$$

Next we build the remaining m edge sets for the output temporal graph. Let $h \in [\log p + m] \setminus [\log p]$ and $j = h - \log p$. The edge set E_h shall later ensure that if a solution to the constructed instance exists, there is also a way to assign the variables of some input instance a_q so that exactly one of the three variables in C_j^q is assigned to *true* (which is why we construct m such layers, one for each clause in a_q).

Let C be the set containing all vertices corresponding to variables that are in clauses $C_j^1,\ldots,C_j^p$, i.e. $C = \bigcup_{q\in[p]} \{v_i^q \in D : x_i^q \in C_j^q\}$. If a solution $S = (P_1,\ldots,P_\tau)$ to the constructed instance exists, it shall be later required that exactly one of the vertices in C is in $V(P_h)$. For this reason we build the following edge set:

$$\text{REQ}_h := \bigcup_{x\in C} \{\{g_0(n,h),x\},\{t,x\}\}$$

Additionally, the vertices in $D \setminus C$ should still be "accessible" (meaning it should be possible for st-paths to include them). To this end, we build an edge set LINE that is similar to LINES above, but it contains only one path (not two as earlier) and it also does not contain an edge to t (as t shall only be "reachable" via the edges in REQ_h):

$$\text{LINE}_h := \{\{s, g_0(1,h)\}\} \cap \bigcup_{i \in [n-1]} \{\{g_0(i,h), g_0(i+1,h)\}\}$$

We also build an edge set ST that is similar to UP and LOW in the first $\log p$ edge sets, but this time we do not need to distinguish between two types. Also it does not contain an edge to t, as every path to t shall only include edges from REQ_h instead.

$$\text{ST}_h := \bigcup_{x \in X_1 \setminus C} \{\{s,x\},\{x,g_0(1,h)\}\} \cup \bigcup_{i \in [n-1]} \bigcup_{x \in X_{i+1} \setminus C} \{\{g_0(i,h),x\},\{x,g_0(i+1,h)\}\}$$

With this we then build the desired edge set E_h :

$$E_h = \text{LINE}_h \cup \text{REQ}_h \cup \text{ST}_h$$

Set $k = 2n$ and $\ell = 2n + 2$. Then $(G = (V, E_1, \ldots, E_{\log p + m}), s, t, k, \ell)$ is the output instance of VERTEX-MSTP, which finishes this construction. ♦

We next show the correctness of Construction 4.12, i.e. that there is a solution of VERTEX-MSTP for the constructed instance of Construction 4.12 if and only if there is a solution to at least one of its input instances $a_1, \ldots, a_p$. The proof is split into one lemma for the forward and one lemma for the backward direction. Again, we will refer to the clause C_j in instance a_q as C_j^q and to the variable x_i in instance a_q as x_i^q.

Lemma 4.13. If at least one of the input instances $a_1, \ldots, a_p$ of Construction 4.12 is a YES-instance of POSITIVE 1-IN-3-SAT, then the output instance is a YES-instance of VERTEX-MSTP.

Proof. If there is a solution assignment to an input instance a_q, let X be the set of those variables that are assigned to *true* in that assignment. Let $Y = \{v_i^q : x_i^q \in X\}$ be the set of vertices corresponding to those variables in X.

First consider any layer G_h, $h \in [\log p]$. There is an st-path of length $n + 2$ formed by LINES_h. As each vertex $v \in Y$ has two neighbours in this path, no two distinct $u, w \in Y$ have the same neighbours and $|Y| = n$, we can extend this path by including the vertices from Y. This way we find an st-path P in G_h, which includes the vertices $Y \cup \{s, t\}$ as well as exactly n others.

Now consider any layer G_h, $h \in [\log p + m] \setminus [\log p]$. Again, LINE_h forms a similar path from s to $g_0(n,h)$, however t is excluded. To reach t from $g_0(n,h)$, exactly one of the vertices from the clauses E_h was built for also has to be included. By definition,

exactly one of those vertices is in Y. The other vertices in Y can extend the path from LINE_h as described above. This means that we find an st-path P' containing $Y \cup \{s,t\}$ as well as exactly n other vertices. As k is set to $2n$ in the output instance, these "remaining vertices" from P and P' may always differ and we found a solution to the output instance this way. □

In order to show the other direction, we first observe the following:

Observation 4.14. If a solution $S = (P_1, \ldots, P_\tau)$ to the constructed instance of Construction 4.12 exists, then $V(P_1) \cap D = \cdots = V(P_\tau) \cap D$.

Proof. Let P_h be an st-path in layer G_h. For each $i \in [n]$ the set $S_i^h = \{g_0(h,i), g_1(h,i)\}$ forms an st-separator of size two in G_h, meaning $|V(P_h) \cap S_i^h| \geq 1$ for each $i \in [n]$. Let S^h be the union of all S_i^h. As all S_i^h are pairwise disjoint, $|V(P_h) \cap S^h| \geq n$.

By construction those vertices get exchanged from layer to layer, i.e. $S^h \cap S^{h+1} = \emptyset$ (in even layers we use the set E' and in odd layers we use the set F'). As $k = 2n$, this means that all other vertices in the st-paths in a solution have to be the same, proving Observation 4.14. □

Any solution to the output instance shall correspond to a solution assignment of exactly one input instance. Thus, Construction 4.12 ensures the following:

Lemma 4.15. If P is an st-path in a solution to the constructed instance of Construction 4.12, then P contains only vertices corresponding to variables of exactly one input instance, that is, there are no two distinct $q, q' \in [p]$ so that $v_i^q \in V(P)$ and $v_{i'}^{q'} \in V(P)$ (for any $i, i' \in [n]$).

Proof. We assume towards a contradiction that there are two vertices $v_i^q, v_{i'}^{q'}$ in $V(P)$ (for any $i, i' \in [n]$) which were constructed for variables of two different input instances a_q and a'_q.

Let Q be the set of vertices corresponding to variables in a_q and let Q' be the set of vertices corresponding to variables in a'_q. As $q \neq q'$, there is a binary position h where they are not similar, i.e. q has a $\mathsf{0}$ at binary position h and q' has a $\mathsf{1}$ at binary position h or vice-versa. This means in the construced layer G_h the vertices in Q and those in Q' are distributed to different sets S_0^h, S_1^h (as named in the construction) and accordingly connected in different edge sets $\text{UP}_h, \text{LOW}_h$. The only two vertices that are connected in UP_h as well as in LOW_h are s and t. Thus, there can be no st-path including vertices connected in UP_h as well as vertices connected in LOW_h, disregarding s and t. This contradicts to P including both v_i^q and $v_{i'}^{q'}$, which was the initial assumption. □

Combining Observation 4.14 and Lemma 4.15 we can now prove the remaing direction for the correctness of Construction 2.16:

Lemma 4.16. If the output instance of Construction 4.12 is a yes-instance of Vertex-MstP, then one of its input instances $a_1, \ldots, a_p$ is a yes-instance of Positive 1-in-3-SAT.

Proof. Let S be a solution to the output instance. Then all paths in S include vertices from D which correspond to variables of the same input instance a_q (Lemma 4.15) and those vertices from D are the same over all st-paths in S (Observation 4.14). Let P be such an st-path in S. Similar to Construction 2.16 the vertices in $V(P) \cap D$ reflect which variables of a_q are assigned to *true* in a solution assignment. Regarding the construction of REQ_h for each $h \in [\log p + m] \setminus [\log p]$, we see that for each clause of a_q exactly one of the three corresponding vertices is in $V(P)$. Therefore we conclude that there is a solution assignment for a_q. □

Having shown the correctness of Construction 4.12, we can now prove that Construction 4.12 satisfies all requirements of being an or-cross-composition:

Lemma 4.17. Construction 4.12 is an or-cross-composition from Positive 1-in-3-SAT into Vertex-MstP parameterised by $\nu + \tau + k + \ell$, where ν is the vertex cover number of the underlying graph of the input temporal graph, τ is the number of layers in the input and k and ℓ are the respective parameters of the input instance.

Proof. We already know that the output instance of Construction 4.12 is a yes-instance of Vertex-MstP if and only if at least one of its input instances is a yes-instance of Positive 1-in-3-SAT as well (Lemmata 4.13 and 4.16).

We also see that Construction 4.12 can be executed time polynomial in $\sum_{q \in [p]} |a_q|$, as for each variable in each input instance there is one vertex created, among $4n + 2$ others, and there are $\tau = \log p + m$ layers. Hence, the amount of temporal edges in the constructed temporal graph G is at most $(pn + 4n + 2)^2 \tau = (pn + 4n + 2)^2 (\log p + m)$.

We note that each constructed temporal edge has an endpoint in $V \setminus D$. This set thus forms a vertex cover of size $4n + 2$ in each constructed layer and consequently also in the underlying graph $G_\downarrow$. We finally note that Construction 4.12 always sets the parameter k to $2n$ and ℓ to $2n + 2$. This means that all regarded parameters (vertex cover number of $G_\downarrow$, number of layers of G, as well as k and ℓ) are polynomially upper-bounded in $\max_{q \in [p]} |a_q| + \log p$. □

With this, Theorem 4.8 follows directly:

Proof of Theorem 4.8. Since there is an or-cross-composition from Positive 1-in-3-SAT to Vertex-MstP parameterised by $\nu + \tau + k + \ell$ (Lemma 4.17), together with Theorem 4.11 we derive that Vertex-MstP does not admit a polynomial kernelisation or polynomial compression for $\nu + \tau + k + \ell$, unless $\mathsf{coNP} \subseteq \mathsf{NP/poly}$. □

4.3 FPT-Algorithm for Multistage st-Path parameterised by Δ and ℓ

In this section, we show that Multistage st-Path is in FPT when parameterised by the maximal degree Δ over all input layers combined with the maximal path length ℓ:

Theorem 4.18: FPT-Algorithm for Vertex-MstP, Edge-MstP and Leven-MstP parameterised by Δ and ℓ

Vertex-MstP, Edge-MstP and Leven-MstP can be decided in $O(\Delta^{2\ell}\tau|V|^2)$ time on an instance $(G=(V,E),s,t,k,\ell)$, where $\Delta = \max_{i\in[\tau]}(\max_{v\in V}(\deg_{G_i}(v)))$.

We will prove Theorem 4.18 by presenting a simple suitable FPT-algorithm. All our considered distance functions, i.e. vertex distance, edge distance and Levenshtein distance, can be computed in quadratic time. Hence, we can use similar algorithms for deciding all three problems which only differ in the computed distance function and, for that reason, refer to any one of those distances only as dist in the following algorithm description. We also assume from now on that there is an st-path of length at most ℓ in each input layer (which can be easily checked in polynomial time) in order to simplify further explanation.

Our algorithms base on the fact that the k' shortest st-paths in a simple graph $G'=(V,E')$ can be found in time polynomial in $k'+|V|$. This has been shown by Yen (1971) and improved by Katoh, Ibaraki, and Mine (1982) for edge-weighted graphs (meaning the length of a path is the sum of its included edges lengths in their setting), the latter formulating an algorithm of running time in $O(k'|V|^2)$. Despite the fact that our setting is much simpler, most notably because there are no edge weights in our initial scenario, we can still use this algorithm by assigning a weight of one to all edges. Since its running time is also sufficiently low, we will from now on assume that the k' shortest st-paths in a simple graph are computed in $O(k'|V|^2)$ time.

That being said, the following Algorithm 4, as stated in pseudo-code, essentially finds all st-paths of length at most ℓ in each input layer (whose number is upper-bounded by Δ^ℓ). It then compares the distance of computed adjacent st-paths according to Definition 1.1 of Multistage st-Path and thus trivially checks if a solution for the input instance can exist.

We now show that Algorithm 4 correctly decides each input instance of Multistage st-Path. In order to do so, we first note that its sub-procedure `FindShortestPaths` is correct:

Observation 4.19. Let $G'=(V,E')$ be a simple graph of maximal degree at most Δ and let $s,t\in V$. Then the call of sub-procedure `FindShortestPaths` in Algorithm 4 with arguments G', s, t, Δ^ℓ and ℓ returns exactly the set of all st-paths with length at most ℓ in G'.

Algorithm 4: The procedure `MstPath` solves MULTISTAGE ST-PATH with a distance function dist in $O(\Delta^{2\ell}\tau|V|^2)$ time, assuming $\text{dist}(p, p')$ can be computed in $O(|V|^2)$ time for two paths p, p' of length at most ℓ. The algorithm relies on the fact that the k' shortest st-paths in a simple graph $G' = (V, E')$ can be found in $O(k'|V|^2)$ time.

```
1  Procedure FindShortestPaths(G' = (V, E'), s, t, k', ℓ)
       Input  : A simple graph G' = (V, E'), two special vertices s, t ∈ V and two
                integers k', ℓ ∈ ℕ.
       Output : A set containing up to k' shortest st-paths in G', which all are of
                length at most ℓ.
2      R ← set of k' shortest st-paths in G'
3      return {p ∈ R : |V(p)| ≤ ℓ}
4  Procedure MstPath(I = (G, s, t, k, ℓ))
       Input  : An instance I of MULTISTAGE ST-PATH, consisting of a temporal
                graph G = (V, E_1, ..., E_τ), two special vertices s, t ∈ V, and two
                integers k, ℓ ∈ ℕ.
       Output : YES, if I is a yes-instance of MULTISTAGE ST-PATH with distance
                function dist. NO otherwise.
5      Δ ← max_{i∈[τ]}(max_{v∈V}(deg_{G_i}(v)))
6      P^1 ← FindShortestPaths((V, E_1), s, t, Δ^ℓ, ℓ)
7      for i ← 2 to τ do
8          P^i ← FindShortestPaths((V, E_i), s, t, Δ^ℓ, ℓ)
9          foreach p ∈ P^i do
10             if there is no p' ∈ P^{i-1} so that dist(p', p) ≤ k then
11                 delete p from P^i
12             end
13         end
14         if P^i = ∅ then
15             return NO
16         end
17     end
18     return YES
```

Proof. Since the maximal degree of G' is at most Δ, G' contains at most Δ^ℓ different st-paths of length at most ℓ. Then all of them are found as subset of $\mathcal{R}$ in the sub-procedure `FindShortestPaths` of Algorithm 4 (line 2), since this sub-procedure is always called to find the Δ^ℓ shortest st-paths in G'. As `FindShortestPaths` then "filters" all paths longer than ℓ from $\mathcal{R}$, the output set contains exactly all st-paths of length at most ℓ in G'. □

Next the correctness of the main procedure of Algorithm 4 can be proven as well:

Lemma 4.20. The procedure `MstPath` in Algorithm 4 returns YES if and only if its input instance I is a YES-instance of MULTISTAGE ST-PATH with distance function dist .

Proof. ($\Leftarrow$) Let $S = (P_1, \ldots, P_\tau)$ be a corresponding solution for I. Then by Observation 4.19 P_1 is in the computed set $\mathcal{P}^1$ (line 6 of Algorithm 4) and each subsequent P_i in S is in the computed set $\mathcal{P}^i$ (line 8 of Algorithm 4). Hence, the computed set $\mathcal{P}^i$ for each $i \in [2, \tau]$ in Algorithm 4 contains a path which does not get deleted for lines 10 to 12 of Algorithm 4, since S is a solution. Since Algorithm 4 only returns NO if some of those sets gets empty, that is, if every element in some set $\mathcal{P}^i$, $i \in [2, \tau]$ gets deleted for lines 10 to 12 of Algorithm 4, the procedure `MstPath` returns YES for I.

($\Rightarrow$) Assume the procedure `MstPath` returned YES. Then the computed set $\mathcal{P}^\tau$ in Algorithm 4 was non-empty when checked at the end (line 14 to 16). Let $P_\tau \in \mathcal{P}^\tau$. Rewinding to lines 9 to 13 of Algorithm 4, we see that there is a $P_{i-1} \in \mathcal{P}^{i-1}$ so that $\text{dist}(P_{i-1}, P_i) \leq k$, as P_τ has not been deleted. Similarly, by backwards induction over these computed sets we infer that for each $i \in [\tau - 1]$ there is a path P_i in the computed set $\mathcal{P}^i$, such that $\text{dist}(P_i, P_{i+1}) \leq k$. Since all these paths are st-paths of length at most ℓ by Observation 4.19, they form a solution of MULTISTAGE ST-PATH with distance function dist for I. □

We are now set to prove Theorem 4.18:

Proof of Theorem 4.18. For each $i \in [\tau]$, the main procedure `MstPath` of Algorithm 4 calls its subprocedure `FindShortestPaths` once, which returns a set P^i of size at most $O(\Delta^\ell)$, taking time in $O(\Delta^\ell |V|^2)$. Each path in P^i then gets compared to each path in P^{i-1} by computing the distance function dist, unless $i = 1$. This step thus takes time in $O(\Delta^{2\ell} |V|^2)$.

Altogether, this results in a running time of Algorithm 4 in $O(\Delta^{2\ell} |V|^2 \tau)$, which is fixed-parameter-tractable for the combined parameter $\Delta + \ell$. Since Algorithm 4 also correctly decides MULTISTAGE ST-PATH by Lemma 4.20, it proves Theorem 4.18. □

4.4 Problem Kernel for Vertex-MstP of Size Polynomial in Feedback Edge Number and Number of Layers

In this section we show that Vertex-MstP admits a polynomial problem kernel when parameterised by the feedback edge number ρ of the underlying graph of the input temporal graph combined with the number of layers τ.

Theorem 4.21

Vertex-MstP admits a problem kernel with size polynomial in $\rho + \tau$, where ρ is the feedback edge number of the underlying graph of the input temporal graph and τ is the number of layers in the input temporal graph.

We prove Theorem 4.21 at the end of this section by presenting a way to kernelise any input instance of Vertex-MstP to a size polynomial in $\rho + \tau$. To this end, we will next describe some data reduction rules for Vertex-MstP.

4.4.1 Data Reductions

In order to reduce vertices of degree one in all input layers, we will apply the following data reduction:

Data Reduction 4.22: If $\deg(v) = 1$ for a vertex $v \in V \setminus \{s, t\}$ in any layer G_i, then delete the edge incident to v in G_i.

Lemma 4.23. Data Reduction 4.22 is *safe* and can be applied exhaustively in $O(\tau n)$ time, where $n = |V|$.

Proof. If $\deg(v) = 1$ in G_i and v is neither s nor t, there there can be no st-path containing v in G_i. Therefore v can be isolated in G_i without changing any solution of the instance. We apply Data Reduction 4.22 exhaustively by checking each $v \in V \setminus s, t$ for its degree once in each layer G_i. If $\deg(v) = 1$ in G_i, then let w be the neighbour of v in G_i. We delete the edge between v and w in G_i and then check if w is now of degree one. If yes, we recursively repeat this process on w (delete its incident edge, check the degree of its neighbour).

The total running time is then $O(\tau n)$, as there are at most n temporal edge deletions and consequently at most $2n$ degree checks done in each layer. □

If there is a vertex which has no neighbours in any input layer, which may for example be a result of the repeated application of Data Reduction 4.22, then we can safely delete this vertex:

Data Reduction 4.24: If $\deg(v) = 0$ for a vertex v in $G_\downarrow$, then delete v from V.

Lemma 4.25. Data Reduction 4.24 is *safe* and can be applied exhaustively in $O(|V|)$ time.

Proof. If v has no edges within any layer, then obviously there can be no solution containing an st-path that includes v. Therefore, v may be deleted without changing any possible solution. □

For the next data reduction, we generalise Multistage st-Path to a weighted problem with a weighted distance function.

Definition 4.26: Weighted Multistage st-Path

Input: A temporal graph $G = (V, E_1, \ldots, E_\tau)$, a weight function $w : V \to \mathbb{N}$, two special vertices $s, t \in V$ and two integers $k, \ell \in \mathbb{N}$.
Question: Are there paths $(P_1, \ldots, P_\tau)$ so that for all $i \in [\tau] : P_i$ connects s to t in $G_i = (V, E_i)$, $\sum_{v \in V(P_i)} w(v) \leq \ell$ and $\text{dist}_w(P_i, P_{i+1}) \leq k$?

Again, dist_w can be any function that somehow measures a distance between two such paths. However, this time it can make use of the weights assigned by the function w. Thus, we will now define dist as a weighted version of our previous vertex distance:

$$\text{dist}_w(P_x, P_y) = \sum_{v \in (V(P_x) \triangle V(P_y))} w(v)$$

We will abbreviate Weighted Multistage st-Path with this weighted vertex distance from now on as Vertex-WMstP. Obviously, an instance of Vertex-MstP can be translated into an equivalent instance of Vertex-WMstP by setting $w(v) = 1$ for each $v \in V$. But now we can additionally define another data reduction for this weighted problem variant:

Data Reduction 4.27: Let $u, v \in V \setminus \{s, t\}$ be two vertices which are adjacent in $G_\downarrow$ and are both of degree two in $G_\downarrow$. Let r be the neighbour of v in $G_\downarrow$ that is not u. Then increase $w(u)$ by $w(v)$ and add the temporal edge $\{r, u\}$ in each layer where v is adjacent to u and r. Delete v and all its temporal edges.

Lemma 4.28. Data Reduction 4.24 is *safe* and can be applied exhaustively in $O(\tau|V|)$ time.

Proof. As u and v are adjacent in $G_\downarrow$ and have both a degree of two, each st-path including u (in any layer) also includes v and vice-versa. Then those two can be "merged"

to one vertex, so that each path including both of them corresponds to exactly one path only including the merged vertex afterwards. In the defined Vertex-WMstP problem, the original paths including u, v and r have the same lengths and distances as the corresponding "new" paths via only u and r with accordingly increased weight. Thus, there is a solution of Vertex-WMstP after applying Data Reduction 4.27 if and only if there was a solution beforehand.

Applying Data Reduction 4.27 once on a vertex v takes $O(\tau)$ time as there has to be a check and a merge operation (with constant time each) within each layer. Applying it on every $v \in V$ hence takes $O(\tau|V|)$ time. □

Using these data reductions, we prove the following:

Lemma 4.29. For any instance I of Vertex-MstP with feedback edge number ρ in $G_\downarrow$ one can compute an equivalent instance I' of Vertex-WMstP in polynomial time, so that the number of vertices of I' is linear in ρ and the number of temporal edges of I' is linear in $\rho\tau$.

Proof. Let $I = (G = (V, E_1, \dots, E_\tau), s, t, k, \ell)$ be an instance of Vertex-MstP with feedback edge number ρ in $G_\downarrow$. Let F be a feedback edge set in $G_\downarrow$ with $|F| = \rho$. We assume that there is an st-path in each layer G_i of G. Otherwise I is a trivial no-instance (which can be recognized in $O(\tau(|V| + |E|))$ time via breadth-first-search in each layer).

We exhaustively apply Data Reductions 4.22 and 4.24 in that order. Then there are no vertices of degree 0 or 1 left in $G_\downarrow$ (except possibly s and t which may both be of degree 1). Afterwards, we "convert" I to an equivalent instance $I' = (G' = (V', E'_1, \dots, E'_\tau), w, s, t, k, \ell)$ of Vertex-WMstP (by setting all vertex weights to one) and then apply Data Reduction 4.27 exhaustively.

Now consider the subgraph T of $G_\downarrow$, which contains exactly all vertices and edges of $G_\downarrow$, except for the edges in F. Then T is a forest. We call vertices of degree one in T *leaves* and vertices of degree at least three in T *branches*. Then each leaf is either in $\{s, t\}$ or is incident to at least one edge in F (otherwise it would have been removed by Data Reduction 4.22 together with Data Reduction 4.24). This means there are at most $2\rho + 2$ many leaves. Each non-leaf vertex is either incident to at least one edge in F, a branch in T, or of degree two in $G_\downarrow$.

As T is a forest and the number of leaves is thus at least the number of branches we infer that the number of branches also has to be at most 2ρ. Due to Data Reduction 4.27 there is only one vertex of degree two per branch or leaf in $G_\downarrow$, that is, we have no paths including consecutive vertices of degree two (disregarding s and t). Hence, the number of vertices in I' is at most $8\rho + 2$. Since T is a forest, it directly follows that T includes at most $8\rho + 1$ edges. Together with the edges in F, we conclude that $G_\downarrow$ includes at most $9\rho + 1$ edges and, consequently, the number of temporal edges in I' is upper-bounded by $9\rho\tau + \tau$. □

4.4.2 Reducing Weights on Vertex-WMstP

By Lemma 4.29 we can convert each instance I of Vertex-MstP to an instance I' of Vertex-WMstP so that its number of vertices is linear in ρ, its number of edges is linear in $\rho\tau$, and I' is equivalent to I. However, the weights assigned by function w in the instance I' are not upper-bounded in ρ. For this reason we have not yet obtained the desired polynomial kernel for Theorem 4.21. Van Bevern, Fluschnik, and Tsidulko (2018) have shown a way how to reduce weights in a weighted variant of an originally unweighted problem, which then leads to a polynomial kernelisation for the original problem (building on previous results for polynomial kernels e.g. by Marx and Végh (2015) and Etscheid et al. (2017)). We are going to take a very similar approach for eventually kernelising Vertex-MstP. In order to do so, we apply the following theorem proven by Frank and Tardos (1987):

Proposition 4.30. There is an algorithm that, on input $w \in \mathbb{Q}^d$ and integer N, computes in polynomial time a vector $w^* \in \mathbb{Z}^d$ with $\|w^*\|_\infty \leq 2^{4d^3} N^{d(d+2)}$ such that $\operatorname{sign}(w \cdot b) = \operatorname{sign}(w' \cdot b)$ for all $b \in \mathbb{Z}^d$ with $\|b\|_1 \leq N - 1$, where

$$\operatorname{sign}(x) = \begin{cases} +1 & \text{for } x > 0 \\ 0 & \text{for } x = 0 \\ -1 & \text{for } x < 0 \end{cases}$$

We later use this algorithm to "shrink" the weights appearing in our instance I' of Vertex-WMstP. Since each weight in an instance of Vertex-WMstP has to be a natural number, we especially note that no weight gets shrinked to a negative value in this process:

Observation 4.31. For $N \geq 2$, Proposition 4.30 maintains the signs of weights by computing w^* from w, i.e. $\operatorname{sign}(w_i) = \operatorname{sign}(w_i^*)$ for all $i \in [d]$.

Proof. For each $i \in [d]$ let e_i by the i-th unit vector (the vector with $e_i = 1$ and $e_j = 0$ for all $j \in [d] \setminus \{i\}$). As $\operatorname{sign}(w \cdot e_i) = \operatorname{sign}(w^* \cdot e_i)$ according to Proposition 4.30 we see that $\operatorname{sign}(w_i) = \operatorname{sign}(w_i^*)$. □

Using Proposition 4.30 and Observation 4.31 on Vertex-WMstP we prove the following:

Lemma 4.32. An instance $I' = (G' = (V', E'_1, \dots, E'_\tau), w, s, t, k, \ell)$ of Vertex-WMstP can be reduced in polynomial time to an instance $I^* = (G', w^*, s, t, k^*, \ell^*)$ of Vertex-WMstP so that

- $w^*(v) \leq 2^{4(|V'|+2)^3}(|V'|+3)^{(|V'|+2)(|V'|+4)}$ for each $v \in V^*$ and
- I^* is a yes-instance if and only if I' is a yes-instance.

Proof. Let $n = |V'|$. In this proof, we conveniently interpret the weight functions w, w^* as vectors in $\mathbb{N}^n$ such that w_v is defined as $w(v)$ for each $v \in V'$ (and similarly for w^*). We apply Proposition 4.30 with $d = n+2$ and $N = n+3$ on the vector (w, k, ℓ) in order to obtain the vector (w^*, k^*, ℓ^*) in polynomial time. Let I', I^* be as stated above. By Proposition 4.30 we know that $\|(w^*, k^*, \ell^*)\|_\infty \leq 2^{4(n+2)^3}(n+3)^{(n+2)(n+4)}$. By Observation 4.31 we also know that (w^*, k^*, ℓ^*) maintains the signs from (w, k, ℓ).

We now show that I^* is a yes-instance if and only if I' is a yes-instance. Let G'_i be a layer in G' and let P_i be an arbitrary st-path in G'_i. Let $x \in \{0,1\}^n$ be a vector so that $x_v = 1$ if and only if $v \in V(P_i)$. Note that $\|(x, 0, -1)\|_1 \leq n+2$. Since $n+2 \leq N-1$, we know by Proposition 4.30 that

$$\operatorname{sign}((w, k, \ell) \cdot (x, 0, -1)) = \operatorname{sign}((w^*, k^*, \ell^*) \cdot (x, 0, -1)).$$

This is equivalent to $\operatorname{sign}(w \cdot x - \ell) = \operatorname{sign}(w^* \cdot x - \ell^*)$ which means that $w \cdot x \leq l \iff w^* \cdot x \leq l^*$. Together with the definition of x we infer that

$$\sum_{v \in V(P_i)} w_v \leq \ell \iff \sum_{v \in V(P_i)} w^*_v \leq \ell^*.$$

This satisfies one of the requirements for I^* to be a yes-instance exactly if I' is a yes-instance, the other being the edge difference condition. Thus let G'_j be another layer in G' and let P_j be an arbitrary st-path in G'_j. Let $y \in \{0,1\}^n$ be a vector so that $y_v = 1$ if and only if $v \in V(P_i) \triangle V(P_j)$. Note that $\|(y, -1, 0)\|_1 \leq n+2$. Since $n+2 \leq N-1$, we know by Proposition 4.30 that

$$\operatorname{sign}((w, k, \ell) \cdot (y, -1, 0)) = \operatorname{sign}((w^*, k^*, \ell^*) \cdot (y, -1, 0)).$$

Together with the definition of y we infer that

$$\sum_{v \in V(P_i) \triangle V(P_j)} w_v \leq k \iff \sum_{v \in V(P_i) \triangle V(P_j)} w^*_v \leq k^*.$$

Thus, I^* is a yes-instance if and only if I' is a yes-instance. □

Particularly, the previous lemma implies that the weight of each vertex in Vertex-WMstP can be reduced so that its encoding length is polynomially upper-bounded in the number of vertices. As we were already able to limit that number to be linear in ρ

(Lemma 4.29), the reduced weights will then also be polynomially upper-bounded in ρ. With this knowledge, we can now finally prove Theorem 4.21:

Proof of Theorem 4.21. Let I be an instance of Vertex-MstP. We then apply Lemma 4.29 and compute an instance I' of Vertex-WMstP with number of vertices in $O(\rho)$ and number of temporal edges in $O(\rho\tau)$, which is equivalent to I. On I' we apply Lemma 4.32 to shrink its vertex weights, such that they are all upper-bounded by $2^{O(\rho^3)}$ and their encoding length is consequently upper-bounded by $O(\rho^3)$. Hence, the total encoding length of this weight-shrinked instance I'' is upper-bounded by $O(\rho^4\tau)$. Vertex-WMstP is, like Vertex-MstP, in NP, since its solutions can be verified in polynomial time and are of size polynomial in the input. For this reason, we can reduce I'' to a new instance I^* of Vertex-MstP in polynomial time. As the size of I'' is polynomially upper-bounded in $\rho+\tau$, so is the size of I^*. This means we found our polynomial kernelisation, which concludes this proof. □

5 Conclusion and Outlook

We introduced and studied Multistage st-path regarding its computational complexity in three different variants, namely Vertex-MstP, Edge-MstP and Leven-MstP. All in all, we found many hardness results in terms of both NP-hardness and parameterised hardness for these problems. Nevertheless, we also spotted that they are fixed-parameter tractible with respect to certain problem parameters.

In the following Section 5.1, we review and discuss the results presented throughout this work. Subsequently, in Section 5.2 we conclude by stating still open questions and future research opportunities which base on the discussed results.

5.1 Discussion

We showed that all studied variants are NP-hard, even if the maximal distance k between adjacent paths is constant, for instance Vertex-MstP is NP-hard even if $k = 0$. For the introductory example this means that even if no sporadically unused office is tolerated, the problem still remains computationally hard to solve. Vertex-MstP and Edge-MstP are also NP-hard if the number of input layers τ is only two, so limiting the amount of considered points in time alone does not make the problem tractable either (unless $\tau = 1$, since then there is no time-dependent part anymore).

In terms of parameterised hardness, we proved that all three variants are W[1]-hard when parameterised by $\tau + k + \ell$ even on bipartite graph layers, where ℓ is the maximal path length. We also showed that Vertex-MstP and Edge-MstP are W[1]-hard when parameterised by $\nu + k + \ell$, where ν is the vertex cover number of the underlying graph of the input temporal graph. The question whether the same W[1]-hardness also holds for Leven-MstP remains open.

As a main result, we then showed that all three researched problem variants admit problem kernels of size exponential in $\tau + \nu$ and are thus fixed-parameter tractable for this combined parameter. To this end, we introduced the concept of temporal twin vertices. We subsequently showed how to efficiently reduce these vertices to a number exponentially upper-bounded in $\tau + \nu$ without changing whether a solution of Multistage st-Path exists. Possibly similar kernelisation methods can be used for other problems also defined on temporal graphs in future research.

However, Vertex-MstP does not admit a kernel of size polynomial in $\tau + \nu + k + \ell$ (unless coNP $\subseteq$ NP/poly). This result suggests that Vertex-MstP is computationally even harder with respect to the parameter $\nu + \tau$ than some other Multistage problems. In particular, Fluschnik et al. (2019) showed that Multistage Vertex Cover admits a problem kernel of size cubic in $\nu + \tau$, despite that the "single-stage" version of their

problem—Vertex Cover—is already NP-complete (which is not the case for Multistage st-Path). It remains open if there is a kernel of size polynomial in $\tau + \nu + k + \ell$ for Edge-MstP or Leven-MstP.

We then showed that our three variants of Multistage st-Path are fixed-parameter tractable with respect to the parameters Δ and ℓ, where Δ is the maximal degree over all input layers. The presented FPT-algorithm exploits the property that there are at most Δ^ℓ *st*-paths of length at most ℓ in each layer. It remains open if there is a corresponding problem kernel of size polynomial in $\Delta + \ell$ for any of the problems.

Lastly, we presented a polynomial-sized kernel for Vertex-MstP with respect to the feedback edge number of the underlying graph $G_\downarrow$ of the input temporal graph combined with the number of layers in the input. This is done by introducing vertex weights to Vertex-MstP (and thus defining a new weighted variant of the problem) and, after applying some data reductions, shrinking those vertex weights by applying a theorem of Frank and Tardos (1987). The resulting new instance of the weighted problem is then equivalent to the original instance and can be reduced in polynomial time to an equivalent instance of Vertex-MstP. However, we doubt that this approach has much practical relevance. In the first place, the feedback edge number of $G_\downarrow$ is often a very large parameter in relation to the input size. Secondly, if one already obtained an equivalent weighted instance of size polynomial in $\rho + \tau$, it appears more reasonable to try solving that weighted instance than to first compute another equivalent instance of Vertex-MstP.

It remains open whether there is an FPT-algorithm or even a kernel of polynomial size for Vertex-MstP with respect to only the feedback edge number of $G_\downarrow$. We presume the first to be true, since each *st*-path in a layer can be constructed by "guessing" a number and order of edges in a minimum feedback edge set and then "filling the gaps" between those ordered feedback edges with non-feedback edges. This is unambiguos as there is at most one path between each two distinct vertices in a forest. Then there has to be some computable function only depending on the feedback edge number of $G_\downarrow$ that upper-bounds the number of *st*-paths in any layer. Accordingly, an algorithm similar to the one in Section 4.3 can probably be employed to solve Vertex-MstP in FPT-time.

5.2 Future Research Opportunities

We already mentioned that some statements proven in this work for Vertex-MstP remain open for Edge-MstP or Leven-MstP. In particular, we raise the question if Edge-MstP or Leven-MstP admits a kernel of size polynomial in $\nu + \tau$, which would make them computationally "easier" with respect to this combined parameter than Vertex-MstP. It would further be interesting to know if there is some structural parameter χ of the input instance, so that Edge-MstP is in FPT with respect to χ, but Vertex-MstP is W[1]-hard for χ (or vice-versa).

Apart from that, future research may clarify

- whether Vertex-MstP is in FPT when parameterised by $\varrho + \tau$, where ϱ is the feedback vertex number of the underlying graph of the input temporal graph,

- whether Vertex-MstP is solvable in polynomial time if $\tau = 2$ and $k = 0$, or
- whether there is a kernel of size polynomial in $\nu + \tau$ for Vertex-MstP if $k = 0$.

The first of these questions seems interesting, since we found a kernel for Vertex-MstP of size exponential in $\nu + \tau$, which is a larger parameter than $\varrho + \tau$. The second question originates from the fact that we established NP-hardness of Vertex-MstP individually for both $\tau = 2$ and $k = 0$, but never combined these restrictions. The third question follows from a similar idea, perhaps it is beneficial to research Vertex-MstP with $k = 0$ as a unique problem in order to find out if it is "easier" than the general case.

Overall, it appears useful to study Multistage st-Path on more restricted input instances. For example, in our described reductions we assumed that temporal edges may change arbitrarily from one layer to its next layer. Assuming the number of "changing edges" $|E(G_i) \triangle E(G_{i+1})|$ to be upper-bounded by some $r \in \mathbb{N}^+$ for each two adjacent layers G_i, G_{i+1} in the input, is Vertex-MstP, Edge-MstP or Leven-MstP fixed-parameter tractable with respect to r (or to $r + \tau$)? For our introductory example, these settings seem plausible, since we can expect e.g. the number of newly-blocked roads to be limited at each point in time.

As a final remark, we also did not study situations where the underlying graph of the input temporal graph is a stronger restricted graph class. For instance, it is open whether Vertex-MstP is fixed-parameter tractable for parameters τ, k, and ℓ if the underlying graph in the input is bipartite.

Bibliography

Ahuja, Ravindra K., James B. Orlin, Stefano Pallottino, and Maria G. Scutellà. 2003. "Dynamic shortest paths minimizing travel times and costs." *Networks* 41, no. 4 (July 1): 197–205. doi:10.1002/net.10072.

An, Hyung-Chan, Ashkan Norouzi-Fard, and Ola Svensson. 2017. "Dynamic Facility Location via Exponential Clocks." *ACM Transactions on Algorithms (TALG)* 13, no. 2 (May 29): 21. doi:10.1145/2928272.

Bampis, Evripidis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. 2018. "Multistage Matchings." In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018),* 101:7:1–7:13. Leibniz International Proceedings in Informatics (LIPIcs). Malmo, Sweden, June. doi:10.4230/LIPIcs.SWAT.2018.7.

Bampis, Evripidis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller. 2019. "Online Multistage Subset Maximization Problems." *arXiv:1905.04162 [cs]* (May 10).

Van Bevern, René, Till Fluschnik, and Oxana Yu Tsidulko. 2018. "Parameterized algorithms and data reduction for the short secluded s-t-path problem." *arXiv:1806.09540 [cs]* (June 25).

Bodlaender, H. L., and B. van Antwerpen - de Fluiter. 2001. "Parallel Algorithms for Series Parallel Graphs and Graphs with Treewidth Two1." *Algorithmica* 29, no. 4 (April): 534–559. doi:10.1007/s004530010070.

Bodlaender, Hans L., Bart M. P. Jansen, and Stefan Kratsch. 2014. "Kernelization Lower Bounds by Cross-Composition" [in en]. *SIAM Journal on Discrete Mathematics* 28, no. 1 (January): 277–305. doi:10.1137/120880240.

Duffin, R.J. 1965. "Topology of series-parallel networks." *Journal of Mathematical Analysis and Applications* 10, no. 2 (April): 303–318. doi:10.1016/0022-247X(65)90125-3.

Eisenstat, David, Claire Mathieu, and Nicolas Schabanel. 2014. "Facility Location in Evolving Metrics." In *Automata, Languages, and Programming,* 459–470. Lecture Notes in Computer Science. Springer Berlin Heidelberg. doi:10.1007/978-3-662-43951-7.

Etscheid, Michael, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. 2017. "Polynomial kernels for weighted problems." *Journal of Computer and System Sciences* 84 (March 1): 1–10. doi:10.1016/j.jcss.2016.06.004.

Fellows, Michael R., Danny Hermelin, Frances Rosamond, and Stéphane Vialette. 2009. "On the parameterized complexity of multiple-interval graph problems." *Theoretical Computer Science* 410, no. 1 (January): 53–61. doi:10.1016/j.tcs.2008.09.065.

Fellows, Michael R., Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. 2008. "Graph Layout Problems Parameterized by Vertex Cover." In *Algorithms and Computation,* 5369:294–305. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-92182-0_28.

Fiala, Jiří, Petr A. Golovach, and Jan Kratochvíl. 2011. "Parameterized complexity of coloring problems: Treewidth versus vertex cover." *Theoretical Computer Science* 412, no. 23 (May): 2513–2523. doi:10.1016/j.tcs.2010.10.043.

Fluschnik, Till, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. 2019. "Multistage Vertex Cover." *arXiv:1906.00659 [cs]* (June 3).

Frank, András, and Éva Tardos. 1987. "An application of simultaneous diophantine approximation in combinatorial optimization." *Combinatorica* 7, no. 1 (March 1): 49–65. doi:10.1007/BF02579200.

Gupta, Anupam, Kunal Talwar, and Udi Wieder. 2014. "Changing Bases: Multistage Optimization for Matroids and Matchings." In *Automata, Languages, and Programming,* 563–575. Lecture Notes in Computer Science. Springer Berlin Heidelberg. doi:10.1007/978-3-662-43948-7.

Hammer, Peter L., and Frédéric Maffray. 1990. "Completely separable graphs." *Discrete Applied Mathematics* 27, no. 1 (May 1): 85–99. doi:10.1016/0166-218X(90)90131-U.

Hassin, Refael. 1992. "Approximation Schemes for the Restricted Shortest Path Problem." *Mathematics of Operations Research* 17, no. 1 (February 1): 36–42. doi:10.1287/moor.17.1.36.

Hernando, C., M. Mora, I. M. Pelayo, C. Seara, and D. R. Wood. 2007. "Extremal Graph Theory for Metric Dimension and Diameter." *Electronic Notes in Discrete Mathematics,* European Conference on Combinatorics, Graph Theory and Applications, 29 (August 15): 339–343. doi:10.1016/j.endm.2007.07.058.

Impagliazzo, R., and R. Paturi. 1999. "Complexity of k-SAT." In *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference) (Cat.No.99CB36317),* 237–240. May. doi:10.1109/CCC.1999.766282.

Karp, Richard M. 1972. "Reducibility among Combinatorial Problems." In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations,* 85–103. The IBM Research Symposia Series. Boston, MA: Springer US. doi:10.1007/978-1-4684-2001-2_9.

Katoh, N., T. Ibaraki, and H. Mine. 1982. "An efficient algorithm for K shortest simple paths." *Networks* 12 (4): 411–427. doi:10.1002/net.3230120406.

Kratsch, Stefan. 2014. "Recent developments in kernelization: A survey." *Bulletin of EATCS* 2 (113).

Malandraki, Chryssi, and Mark S. Daskin. 1992. "Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms." 26, no. 3 (August 1): 185–200. doi:10.1287/trsc.26.3.185.

Marx, Dániel, and László A. Végh. 2015. "Fixed-Parameter Algorithms for Minimum-Cost Edge-Connectivity Augmentation." *ACM Trans. Algorithms* 11, no. 4 (April): 27:1–27:24. doi:10.1145/2700210.

Masek, William J., and Michael S. Paterson. 1980. "A faster algorithm computing string edit distances." *Journal of Computer and System Sciences* 20, no. 1 (February): 18–31. doi:10.1016/0022-0000(80)90002-1.

Muzi, Irene, Michael P. O'Brien, Felix Reidl, and Blair D. Sullivan. 2017. "Being Even Slightly Shallow Makes Life Hard." In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, 83:79:1–79:13. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany. doi:10.4230/LIPIcs.MFCS.2017.79.

Schaefer, Thomas J. 1978. "The complexity of satisfiability problems." In *Proceedings of the tenth annual ACM symposium on Theory of computing - STOC '78*, 216–226. San Diego, California, United States: ACM Press. doi:10.1145/800133.804350.

Wu, Huanhuan, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. "Path problems in temporal graphs." *Proceedings of the VLDB Endowment* 7, no. 9 (May 1): 721–732. doi:10.14778/2732939.2732945.

Yen, Jin Y. 1971. "Finding the K Shortest Loopless Paths in a Network." *Management Science* 17 (11): 712–716.

Yu, Gang, and Jian Yang. 1998. "On the Robust Shortest Path Problem." *Computers & Operations Research* 25, no. 6 (June): 457–468. doi:10.1016/S0305-0548(97)00085-3.

Yujian, L., and L. Bo. 2007. "A Normalized Levenshtein Distance Metric." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, no. 6 (June): 1091–1095. doi:10.1109/TPAMI.2007.1078.