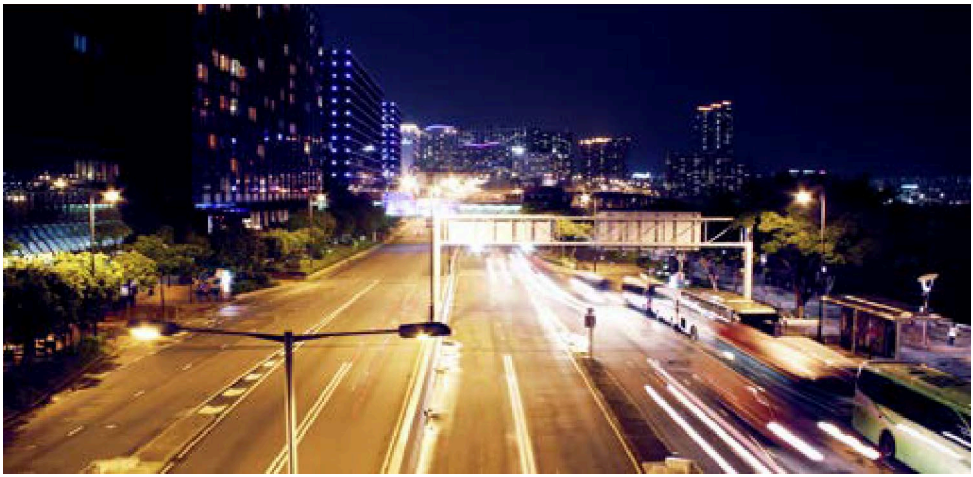
Foundations of computing Volume 13

Hendrik Molter

Classic Graph Problems Made Temporal – A Parameterized Complexity Analysis



Universitätsverlag der TU Berlin

Technische Universität Berlin

Hendrik Molter

Classic Graph Problems Made Temporal – A Parameterized Complexity Analysis

The scientific series *Foundations of computing* of the Technische Universität Berlin is edited by:
Prof. Dr. Stephan Kreutzer,
Prof. Dr. Uwe Nestmann,
Prof. Dr. Rolf Niedermeier

Foundations of computing | 13

Hendrik Molter

Classic Graph Problems Made Temporal – A Parameterized Complexity Analysis

Universitätsverlag der TU Berlin

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the internet at http://dnb.dnb.de.

Universitätsverlag der TU Berlin, 2020
http://www.verlag.tu-berlin.de

Fasanenstr. 88, 10623 Berlin
Tel.: +49 (0)30 314 76131 / Fax: -76133
E-Mail: publikationen@ub.tu-berlin.de

Zugl.: Berlin, Techn. Univ., Diss., 2019
Gutachter: Prof. Dr. Rolf Niedermeier
Gutachter: Prof. Dr. Thomas Erlebach (University of Leicester)
Gutachter: Dr. Ralf Klasing (Université de Bordeaux)
Die Arbeit wurde am 19. Dezember 2019 an der Fakultät IV unter
Vorsitz von Prof. Dr. Benjamin Blankertz erfolgreich verteidigt.

Cover image: Hendrik Molter, 2011
CC BY 4.0 | http://creativecommons.org/licenses/by/4.0/

Print: docupoint GmbH
Layout/Typesetting: Hendrik Molter

ORCID iD Hendrik Molter: 0000-0002-4590-798X
http://orcid.org/0000-0002-4590-798X

ISBN 978-3-7983-3172-3 (print)
ISBN 978-3-7983-3173-0 (online)

ISSN 2199-5249 (print)
ISSN 2199-5257 (online)

Published online on the institutional repository of the Technische Universität Berlin:
DOI 10.14279/depositonce-10551
http://dx.doi.org/10.14279/depositonce-10551

Zusammenfassung

In dieser Dissertation untersuchen wir die parametrisierte Berechnungskomplexität von sechs klassischen Graphproblemen, die in einen temporalen Kontext gestellt werden. Formaler ausgedrückt betrachten wir Probleme, die auf *temporalen Graphen* definiert sind, wobei temporale Graphen aus einer unveränderlichen Knotenmenge bestehen, zusammen mit einer Kantenmenge, die sich über einen diskreten Zeitraum hinweg verändern darf. Temporale Graphen eignen sich besonders gut zum Modellieren dynamischer Daten und sind daher in Situationen von Bedeutung, in denen dynamische Veränderungen oder zeitabhängige Interaktionen eine wichtige Rolle spielen. Beispiele hierfür sind das Betrachten von Kommunikationsnetzwerken, sozialen Netzwerken oder Netzwerken, deren Interaktionen räumliche Annäherungen modellieren. Das wichtigste Auswahlkriterium für unsere Problemstellungen war, dass sie in Kontexten der Analyse dynamischer Daten wohlmotiviert sind.

Da temporale Graphen mathematisch gesehen komplexer sind als statische Graphen, ist es vielleicht nicht sehr überraschend, dass alle Probleme, die wir in dieser Dissertation betrachten, NP-schwer sind. Wir konzentrieren uns auf die Entwicklung exakter Algorithmen, wobei wir versuchen FPT-Resultate zu erzielen oder durch spezialisierte Reduktionen zu zeigen, dass das betrachtete Problem NP-schwer auf sehr eingeschränkten Instanzen ist, oder berechnungsschwer im parametrisierten Sinne bezüglich möglichst „großer“ Parameter ist. Im Kontext temporaler Graphen betrachten wir in erster Linie Strukturparameter des *unterliegenden Graphen*, das heißt des Graphen, den man erhält, wenn man alle Zeitinformationen ignoriert. Allerdings studieren wir auch andere Parameter, zum Beispiel solche, die das Ausmaß der zeitlichen Veränderung eines temporalen Graphen messen. Im Folgenden geben wir einen kurzen Überblick über unsere Problemstellungen und wichtigsten Ergebnisse.

Restless Temporal Paths. Ein Pfad in einem temporalen Graph muss Kausalität oder *Zeit* respektieren. Dies bedeutet, dass die Kanten, die von dem temporalen Pfad benutzt werden, nicht zu abnehmenden Zeitpunkten erscheinen dürfen. Wir untersuchen temporale Pfade, die darüber hinaus eine maximal erlaubte Wartezeit in allen Knoten haben. Unsere Hauptresultate sind, dass Pfade dieser Art zu finden NP-schwer ist, sogar in sehr restriktiven Instanzen, und dass das Problem W[1]-schwer bezüglich der kreiskritischen Knotenzahl des unterliegenden Graphen ist.

Temporal Separators. Ein temporaler Separator ist eine Knotenmenge, die, wenn sie aus einem temporalen Graph entfernt wird, alle temporalen Pfade zwischen zwei

ausgewählten Knoten zerstört. Wir erzielen hier zwei Hauptresultate: Auf der einen Seite untersuchen wir die Berechnungskomplexität des Findens von temporalen Separatoren in temporalen Einheitsintervallgraphen, einer Verallgemeinerung von Einheitsintervallgraphen im temporalen Kontext. Wir zeigen, dass das Problem auf temporalen Einheitsintervallgraphen NP-schwer ist, aber identifizieren eine weitere Einschränkung, die es erlaubt, das Problem in Polynomzeit zu lösen. Auf Letzterem aufbauend entwickeln wir einen FPT-Algorithmus, der eine „Distanz-zur-Trivialität"-Parametrisierung nutzt. Auf der anderen Seite zeigen wir, dass das Finden temporaler Separatoren, die alle *ruhelosen* temporalen Pfade zerstören, Σ_2^p-schwer ist.

Temporal Matchings. Wir führen ein Modell für Matchings in temporalen Graphen ein, bei dem sich zwei Knoten „erholen" müssen, nachdem sie zu einem bestimmten Zeitpunkt einander zugeordnet wurden. Das heißt, für eine gewisse Zeit können diese Knoten nicht wieder einander zugeordnet werden. Wir nutzen das Konzept von temporalen Kantengraphen, um zu zeigen, dass das Finden von temporalen Matchings NP-schwer ist, selbst wenn der unterliegende Graph ein Pfad ist.

Temporal Coloring. Wir übertragen das klassische Knotenfärbungsproblem in den temporalen Rahmen. In unserem Modell muss jede Kante in jedem Zeitfenster einer bestimmten Größe mindestens einmal gültig gefärbt sein, also beide Endpunkte eine andere Farbe haben. Wir zeigen, dass dieses Problem schon auf sehr eingeschränkten Instanzen NP-schwer ist – sogar für zwei Farben. Wir beschreiben einfache Exponentialzeitalgorithmen für dieses Problem. Eines unserer Hauptresultate ist, dass diese Algorithmen vermutlich nicht signifikant verbessert werden können.

Temporal Cliques and *s*-Plexes. Wir stellen ein Modell für temporale *s*-Plexe vor, das eine kanonische Verallgemeinerung eines existierenden Modells für temporale Cliquen ist. Unser Hauptresultat ist ein FPT-Algorithmus, der alle maximalen temporalen *s*-Plexe aufzählt, wobei wir eine temporale Variante von Degeneriertheit von Graphen als Parameter benutzen.

Temporal Cluster Editing. Wir stellen ein Modell für Clustereditierung in temporalen Graphen vor, bei dem wir alle „Schichten" eines temporalen Graphens in hinreichend ähnliche Cluster überführen wollen. Unsere Hauptergebnisse sind zum einen ein FPT-Algorithmus bezüglich des Parameters „Anzahl der Editierungen plus Ähnlichkeit der Cluster". Zum anderen geben wir eine effiziente Vorverarbeitungsmethode an, welche die Größe der Eingabeinstanz beweisbar so reduziert, dass sie unabhängig von der Anzahl der Knoten der ursprünglichen Instanz ist.

Abstract

This thesis investigates the parameterized computational complexity of six classic graph problems lifted to a temporal setting. More specifically, we consider problems defined on *temporal graphs*, that is, a graph where the edge set may change over a discrete time interval, while the vertex set remains unchanged. Temporal graphs are well-suited to model dynamic data and hence they are naturally motivated in contexts where dynamic changes or time-dependent interactions play an important role, such as, for example, communication networks, social networks, or physical proximity networks. The most important selection criteria for our problems was that they are well-motivated in the context of dynamic data analysis.

Since temporal graphs are mathematically more complex than static graphs, it is maybe not surprising that all problems we consider in this thesis are NP-hard. We focus on the development of exact algorithms, where our goal is to obtain fixed-parameter tractability results, and refined computational hardness reductions that either show NP-hardness for very restricted input instances or parameterized hardness with respect to "large" parameters. In the context of temporal graphs, we mostly consider structural parameters of the *underlying graph*, that is, the graph obtained by ignoring all time information. However, we also consider parameters of other types, such as ones trying to measure how fast the temporal graph changes over time. In the following we briefly discuss the problem setting and the main results.

Restless Temporal Paths. A path in a temporal graph has to respect causality, or *time*, which means that the edges used by a temporal path have to appear at non-decreasing times. We investigate temporal paths that additionally have a maximum waiting time in every vertex of the temporal graph. Our main contributions are establishing NP-hardness for the problem of finding restless temporal paths even in very restricted cases, and showing W[1]-hardness with respect to the feedback vertex number of the underlying graph.

Temporal Separators. A temporal separator is a vertex set that, when removed from the temporal graph, destroys all temporal paths between two dedicated vertices. Our contribution here is twofold: Firstly, we investigate the computational complexity of finding temporal separators in *temporal unit interval graphs*, a generalization of unit interval graphs to the temporal setting. We show that the problem is NP-hard on temporal unit interval graphs but we identify an additional restriction which makes the problem solvable in polynomial time. We use the latter result to develop a fixed-parameter algorithm with a "distance-to-triviality" parameterization.

Secondly, we show that finding temporal separators that destroy all *restless* temporal paths is Σ_2^P-hard.

Temporal Matchings. We introduce a model for matchings in temporal graphs, where, if two vertices are matched at some point in time, then they have to "recharge" afterwards, meaning that they cannot be matched again for a certain number of time steps. In our main result we employ *temporal line graphs* to show that finding matchings is NP-hard even on instances where the underlying graph is a path.

Temporal Coloring. We lift the classic graph coloring problem to the temporal setting. In our model, every edge has to be colored properly (that is, the endpoints are colored differently) at least once in every time interval of a certain length. We show that this problem is NP-hard in very restricted cases, even if we only have two colors. We present simple exponential-time algorithms to solve this problem. As a main contribution, we show that these algorithms presumably cannot be improved significantly.

Temporal Cliques and *s*-Plexes. We propose a model for temporal s-plexes that is a canonical generalization of an existing model for temporal cliques. Our main contribution is a fixed-parameter algorithm that enumerates all maximal temporal s-plexes in a given temporal graph, where we use a temporal adaptation of degeneracy as a parameter.

Temporal Cluster Editing. We present a model for cluster editing in temporal graphs, where we want to edit all "layers" of a temporal graph into cluster graphs that are sufficiently similar. Our main contribution is a fixed-parameter algorithm with respect to the parameter "number of edge modifications" plus the "measure of similarity" of the resulting clusterings. We further show that there is an efficient preprocessing procedure that can provably reduce the size of the input instance to be independent of the number of vertices of the original input instance.

Preface

This thesis is based on some results of my research activity at TU Berlin in the Algorithmics and Computational Complexity group of Rolf Niedermeier from October 2015 until September 2019. I gratefully acknowledge support from the Deutsche Forschungsgesellschaft (DFG) and the TU Berlin. From October 2015 to September 2016, I was financially supported by the DFG, project DAPA (NI 369/12) and from February 2018 up to the time of writing I have been financially supported by the DFG, project MATE (NI 369/17). In the time between those two projects I was financially supported by the TU Berlin.

Since I started, my research quickly focused on temporal graph problems. However, I was also involved in a variety of projects from other research areas within graph algorithmics and complexity theory. Most of my research has been in close collaboration with my coauthors. In the research projects covered in this thesis my coauthors were, in alphabetical order, Matthias Bentert, Arnaud Casteigts, Jiehua Chen, Till Fluschnik, Anne-Sophie Himmel, George B. Mertzios, Marco Morik, Rolf Niedermeier, Malte Renken, René Saitenmacher, Manuel Sorge, Ondřej Suchý, Viktor Zamaraev, and Philipp Zschoche. In the following, I briefly explain my contributions in the respective research projects.

Chapter 3: Restless Temporal Paths. The idea to study Restless Temporal (s,z)-Path emerged as a follow-up work to the Master's thesis of Anne-Sophie Himmel [Him18] and was partially, in the context of separators and temporal walks, already considered in the Master's thesis of Philipp Zschoche [Zsc17], which I cosupervised with Till Fluschnik and Rolf Niedermeier. While we have been interested in this problem for a while and became more interested after discovering its NP-hardness, the project picked up speed when Arnaud Casteigts (Université de Bordeaux) visited our group in May and June 2019. Arnaud was interested in the problem and we jointly developed most of our results during the time of his visit. I was mostly involved in the computational hardness results I present in this thesis, especially the W[1]-hardness for the parameter "feedback vertex number of the underlying graph". At the time of writing of this thesis, the results were available only on ArXiv and were being prepared for a conference publication. At the date of publication of this thesis, additional results have been obtained and the paper is accepted for publication at the *31st International Symposium on Algorithms and Computation (ISAAC '20)* [Cas+20].

Chapter 4: Temporal Separators. This project traces back to the Master's thesis of Philipp Zschoche [Zsc17], which I co-supervised with Till Fluschnik and Rolf Niedermeier. After Philipp completed his thesis, we continued investigating the computational complexity of TEMPORAL (s,z)-SEPARATION and developed two different viewpoints on this problem, splitting the work into two projects. In the first project, we put an emphasis on investigating the difference between separators for strict and non-strict temporal paths [Zsc+18, Zsc+20]. Philipp presented the results at the *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS '18)* [Zsc+18] and we published a long version in the *Journal of Computer and System Sciences* [Zsc+20]. In the second project, we focused on trying to find tractable cases for TEMPORAL (s,z)-SEPARATION by investigating the computational complexity of the problem on restricted classes of temporal graphs [Flu+18, Flu+20b]. Philipp presented the results at the *44th International Workshop of Graph-Theoretic Concepts in Computer Science (WG '18)* [Flu+18] and we published an extended version in the journal *Theoretical Computer Science* [Flu+20b] with additional input from Malte Renken. While many results in both projects were jointly developed by all authors, my main contribution was the investigation of TEMPORAL (s,z)-SEPARATION on temporal unit interval graphs which was part of the second project [Flu+18, Flu+20b] and which I also feature in this thesis. I was invited to give a talk about our results at the satellite workshop *Algorithmic Aspects of Temporal Graphs* of *ICALP '18*.

I obtained the Σ_2^P-hardness result for RESTLESS TEMPORAL (s,z)-SEPARATION shortly after working on the project on RESTLESS TEMPORAL (s,z)-PATH [Cas+20]. Up until the date of publication of this thesis, this result is exclusively featured in this thesis.

Chapter 5: Temporal Matchings. This project was initiated during a visit of George B. Mertzios and Viktor Zamaraev (then both Durham University) at our research group in December 2018. In this time we developed some of the hardness results and an approximation algorithm for TEMPORAL MATCHING. After this, I tried for a long time to find an FPT-algorithm for structural parameters of the underlying graph such as treewidth or vertex cover number. After a number of unsuccessful tries I was beginning to suspect that this might not be possible and discovered the result showing that TEMPORAL MATCHING is NP-hard even if the underlying graph is a path, which is also my main contribution to this project, and which is also featured in this thesis. At the time of writing this thesis, our results were published on ArXiv and the paper was submitted to a conference and under review. At the date of publication of this thesis, I presented the results at the *37th International Symposium on Theoretical Aspects of Computer Science (STACS '20)* [Mer+20] and a long version of the paper has been submitted to a journal and is under review.

Chapter 6: Temporal Coloring. At a research retreat of our research group in Boiensdorf, Baltic Sea (2017), George B. Mertzios (Durham University) proposed to study Temporal Coloring. After some discussions, we started working on this project shortly after Viktor Zamaraev (Durham University) joined George as a post-doc. Most results were developed jointly, however, I was mainly involved in the computational hardness results. I presented our results at the *33rd AAAI Conference on Artificial Intelligence (AAAI '19)* [MMZ19] and at the time of writing, a journal version was in preparation. At the date of publication of this thesis, a long version of this paper has been submitted to a journal and is under review.

Chapter 7: Temporal Cliques and *s*-Plexes. This project started with the Bachelor's thesis on Temporal Clique of Anne-Sophie Himmel [Him16], which I co-supervised with Manuel Sorge and Rolf Niedermeier. After Anne-Sophie finished her thesis, we published the results and I presented them at the *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM '16)* [Him+16] where we won the runner-up best paper award. A long version appeared at *Social Network Analysis and Mining* [Him+17]. In this first part of the project, I was mostly involved in the theoretical part of developing an FPT-algorithm to enumerate temporal cliques for the parameter "Δ-slice degeneracy".

We followed this work up in a student project with participants Marco Morik and René Saitenmacher, which I co-supervised with Matthias Bentert, where we started to investigate the problem Temporal s-Plex. After the student project finished, we continued this work with the help of Anne-Sophie and Rolf. Anne-Sophie presented our results at the *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM '18)* [Ben+18] and a journal version appears in the *ACM Journal of Experimental Algorithmics* [Ben+19]. In this second project, I was again mainly involved in the theoretical results. Notably, both projects featured an implementation and experimental evaluations which are not presented in this thesis.

Chapter 8: Temporal Cluster Editing. I proposed this topic at a research retreat of our research group in Boiensdorf, Baltic Sea (2017) and was joined by Jiehua Chen, Manuel Sorge (then both University of Warsaw), and Ondřej Suchý (Czech Technical University in Prague). During the period of the research retreat we jointly developed the main ideas for the kernelization algorithm for Temporal Cluster Editing which is featured in this thesis as well as parts of the FPT-algorithm. After the retreat I focused on developing and refining the FPT-algorithm which is also featured in this thesis. At a later point, we started investigating a second problem variant which is not

covered in this thesis. I presented our results at the *29th International Symposium on Algorithms and Computation (ISAAC '18)* [Che+18] and at the time of writing, a journal version is in preparation.

Further Research Projects. In the following, I give a very brief list of projects I was involved in during my research activity at TU Berlin that are not covered in this thesis. The ordering is roughly chronological. I was involved in work on h-index manipulation [Bev+16b, Bev+20], multi-layer subgraph detection [Bre+17, Bre+19], secluded problems [Bev+16a, Bev+18], the minimum shared edges problem [Flu+17], centrality improvement [HMS18], funnels [Mil+18, Mil+20] (a class of directed acyclic graphs), diminishers [Fer+18, Fer+20] (a framework to refute certain polynomial kernels), collapsing k-cores [LMS18], dynamic cluster editing [Luo+18, Luo+20], enumerating isolated temporal cliques [MNR19, MNR20], temporal feedback edge sets [Haa+20], a contribution to a Festschrift in honor of Hans Bodlaender's 60th birthday on temporal treewidth [Flu+20a], and work on temporal betweenness [Buß+20].

Acknowledgements. First of all, I am sincerely grateful to Rolf Niedermeier, who gave me the opportunity to work in his group as a PhD student and contributed a lot of his time for guidance, advice, and support in many different forms that allowed me to finish this thesis. Together with him, I want to thank Thomas Erlebach and Ralf Klasing for reviewing my thesis and their valuable feedback which greatly helped to improve the presentation of my results.

I am grateful to all my coauthors for pleasant and fruitful collaborations without which this thesis would not have been possible. In lexicographical order, I want to thank Matthias Bentert, René van Bevern, Robert Bredereck, Sebastian Buß, Arnaud Casteigts, Jiehua Chen, Henning Fernau, Till Fluschnik, Marcelo Garlet Millani, Roman Haag, Steffen Härtlein, Maike Hatzel, Danny Hermelin, Anne-Sophie Himmel, Clemens Hoffmann, Christian Komusiewicz, Stefan Kratsch, Andreas Krebs, Junjie Luo, George B. Mertzios, Marco Morik, André Nichterlein, Rolf Niedermeier, Malte Renken, Maciej Rymar, René Saitenmacher, Henning Seidler, Manuel Sorge, Ondřej Suchý, Toby Walsh, Viktor Zamaraev, and Philipp Zschoche.

I also want to thank Christlinde Thielcke for helping me with numerous administrative issues and all my colleagues, longterm visitors and friends of our group who I have not collaborated with (I hope we can change that some time!) for good times and many interesting discussions. In lexicographical order: Niclas Böhmer, Markus Brill, Piotr Faliszewski, Vincent Froese, Anne-Marie George, Klaus Heeger, Falk Hüffner, Andrzej Kaczmarczyk, Leon Kellerhals, Dušan Knop, Tomohiro Koana,

Ulrike Schmidt-Kraepelin, Piotr Skowron, Nimrod Talmon, and Mathias Weller.

Special thanks go out to my colleagues Matthias, Till, Vincent, Anne-Sophie, Leon, André, Rolf, Malte, and Philipp as well as my sister Hannah Molter and my father Karl Molter for helping me to improve the presentation of my results, and to Jiehua for providing me with the latex template. I want to give extra special thanks to Till, Anne-Sophie, Hannah, as well as my flatmates Maria Efimova and Johann Zajaczkowski for mental support and encouragement during my write-up phase.

Finally, I want to thank my family Anne, Karl, and Hannah Molter, my extended family and all my friends, and everybody who supported me in one form or another.

Contents

CHAPTER 1

Introduction and Overview

Many questions that we want computers to answer for us are of the form that we have some data set, which is typically too large or complex to be fully investigated by hand, and we want to know some specific fact about this data. This could for example be whether the data fulfills some global property (or how we have to modify it to obtain this property), or whether we can find a small subset in the data that fulfills a specific property (where the latter could be viewed as a special case of the former). To find algorithms that can perform these tasks or to analyze how hard it is for a computer to solve these problems, we typically need an abstract mathematical model of our data first, which, on the one hand, in many cases reduces the amount of information to something which a computer can handle in a reasonable amount of time, and, on the other hand, allows us to formulate our questions in a precise and formal way.

Graphs, that is, sets of *vertices* that are joined together in pairs by *edges*, turned out to be one of the most versatile mathematical models for complex data sets. They are used in a wide variety of research areas. In all kinds of applications related to networks, graphs are typically the underlying mathematical model. Specific tasks that pop up in these research fields are then commonly modeled as graph problems and computer scientists try to find efficient algorithms to solve them (or show that neither they nor many other famous people can find an efficient algorithm).

However, as predicted by Moore's law, the capabilities of computers keep increasing at a stunning pace, and at the same time our understanding of graph problems becomes better and better. Hence, we can dial back our level of abstraction and investigate problems on generalized graph models that incorporate more information of the data. Indeed, this is already happening since several years. In particular, dynamics of interactions play an increasingly important role in the analysis of complex data. Especially data that is usually modeled with graphs, such as for example communication data, is often times inherently changing over time. The classic model of graphs as we know does not reflect this behavior of the data which leads to a loss of information that can be critical in certain problem settings. This subject matter is perhaps best illustrated through the following, somewhat whimsical, example.

Imagine that you, like the author, are part of a fairly large online social network

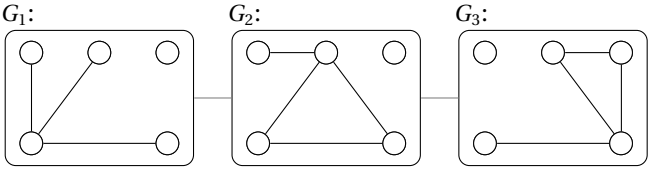


Figure 1.1: Example temporal graph with lifetime three. We see three snapshots of the temporal graph, one for each time step. These graphs G_1, G_2, and G_3 contain all edges that are present at the respective time steps one, two, and three. The graphs G_1, G_2, and G_3 are also called the *layers* of the temporal graph.

and suddenly realize that almost all of your "friends" own a "bandersnatch". And you start wondering where this new trend comes from. Who is manufacturing these bandersnatches? You have not really heard of them before and you did not know that you would need one until now. You start to realize that bandersnatches were mostly promoted by viral marketing and word-of-mouth, and you would like to analyze your social network to trace back this new trend to its origin.

Assume that, for reasons we do not discuss here, you have access to the whole social network. You are one of the last individuals not owning a bandersnatch. Hence, when looking at the data and formalizing the problem, you quickly realize that without information about the times of interactions, it will not be possible to extract any useful information. However, if the interactions between people are time-stamped, then you are able create a much more meaningful model for the dissemination of the bandersnatch.

More formally, we use *temporal graphs* as the main mathematical model to represent our data in this thesis. A temporal graph is, informally speaking, a graph where the edge set may change over a discrete time interval, called the *lifetime*, while the vertex set remains unchanged, see Figure 1.1 for an example. Considering the above example, we can see that much more expressive models for information spreading phenomena can be formulated on temporal graphs than on classic non-temporal graphs.

1.1 Invitation to Temporal Graph Problems

There are many canonical ways to formulate problems on temporal graphs. In the following we discuss which types of problems we consider in this thesis. First of all, we only consider *finite* temporal graphs, that is, temporal graphs with a finite number of vertices and a finite lifetime. Secondly, we always assume that we have

full knowledge of the temporal graph, that is, the whole temporal graph is part of the input, as opposed to an *online* setting, where we would be informed about the changes in the graph when they happen. Third, we assume that we look for a centralized algorithm that can access all parts of the temporal graph, as opposed to a *distributed* setting, where a computing entity can only access information about a limited part of the temporal graph.

The temporal graph problems we consider are, in the most general form, of the type "does the given graph have a certain property?". If we want to be more specific, then we can categorize most of the temporal graph problems we investigate in this thesis into the following two groups, which in an analogous way also exists for classic graph problems.

Substructure detection problems: We want to know whether a given temporal graph contains a substructure that has a certain property. This can be seen as a search problem, where once we find a desired substructure, it does not matter how the rest of the temporal graph looks like. An example from classic graph problems would be CLIQUE, where we want to find a set of vertices which are all pairwise connected.

Graph modification problems: We want to know whether a given temporal graph can be modified such that it (as a whole) fulfills a certain property. Modification in this context typically means adding or removing edges, or removing vertices. An example from classic graph problems would be CLUSTER EDITING, where we want to add to or remove edges from a given graph such that every connected component of the resulting graph is a clique.

Of course there are many natural temporal graph problems which do not fit into any of the two categories.

When it comes to solving temporal graph problems we focus on exact algorithms and perform worst-case running time analyses. Since the temporal dimension adds more complexity to our problems (compared to classic graph problems), it is not surprising that all problems we consider in this thesis are NP-hard. Hence, we presumably have to expect algorithms for our problems to have exponential running times. This motivates a *parameterized* complexity analysis. Here we identify secondary properties of the input instances of our problems (other than the size) that can be quantified with some natural number, which we call the *parameter*. The goal is to find parameters that, informally speaking, allow for efficient algorithm when they are small. Formally, we measure the running time of algorithms in terms of the input size as well as the parameter and aim for running times of the form $f(p) \cdot |I|^{O(1)}$,

where p is the parameter, I is the problem instance, and f is a computable function. In other words, we want to find algorithms that have polynomial running times for constant parameter values, and the parameter should only influence the leading constant of the polynomial and not its degree. If a problem admits such an algorithm, then we call it *fixed-parameter tractable* when parameterized by p. If we cannot find such algorithms, then we aim to show that the existence of such algorithms is unlikely under widely believed computational complexity assumptions.

1.2 Related Work

The theory of temporal graphs and temporal graph algorithmics are comparatively young research fields but have strongly grown in last years. There are several surveys and books available that give an overview on the different research activities in these fields [Boc+14, Cas+12, CF13a, CF13b, Hol15, HS12, HS13, HS19, LVM18, Mic16]. We an extended discussion of related work in each of the chapters of this thesis which is then specific to the problem that we investigate in the respective chapter.

1.3 Thesis Contribution and Overview

As the title of this thesis already describes, we take a look at temporal versions of several classic graph problems. In most cases, there were already temporal models of the problems established in the literature, and in some cases we also propose how to transfer the problems to the temporal setting. We selected the problem by the following two main considerations.

1. We look at problems that are naturally motivated in a temporal context, much as in our introductory example. For many classic graph problems appearing in data science or network analysis it is very canonical to investigate them in a temporal setting.

2. As a general rule of thumb we can say that transferring a problem to the temporal setting makes it more complicated. Because of that we prefer to investigate problems that are efficiently solvable in the classic setting and we do *not* want to look at problems that are already very difficult or not very well understood in the classic setting.

We investigate temporal versions of six classic graph problems, each of which is discussed in one chapter of this thesis. This thesis starts with a short presentation of the most important concepts from (temporal) graph theory and parameterized complexity theory in Chapter 2. In Chapter 9 the thesis concludes, summarizing

and discussing again the main results, pointing out some general observations, and providing a view on possible future research directions. In the six chapters in between, we present the results we achieved when investigating the respective temporal graph problems. We start with two problems related to connectivity, that is, finding specific types of temporal paths and separators in temporal graphs. Then we investigate a model of matching for temporal graph. Next, we move to the problem of coloring a temporal graph. Finally, we give our findings for two problems related to finding dense subgraphs in a temporal graph, which are enumerating dense subgraphs and clustering the temporal graph into non-overlapping dense subgraphs. In the following, we briefly introduce each problem and give an overview on the most important contributions. For problem motivations and related work we refer to the introductory sections of the corresponding chapters.

Chapter 3: Restless Temporal Paths. We begin by investigating the problem of finding a special type of path in temporal graphs. This problem clearly falls into the "substructure detection" category. Finding paths between two vertices in graphs is arguably one of the most basic and important problems in graph algorithmics and it was also one of the first problems transferred to the temporal setting. A path from a start vertex to a destination vertex in a temporal graph has to respect causality, or *time*, which means that the edges used by a temporal path have to appear at non-decreasing times. Temporal paths have received much attention in recent years and many efficient algorithms are known to find them.

In this chapter we investigate temporal paths that additionally have a maximum waiting time in every vertex of the temporal graph. This means a path has to "continue" after a certain amount of time steps after its arrival at a vertex. This is a very natural and well-motivated restriction, however, this makes the problem of finding paths in a temporal graph computationally hard. Our main contribution is to thoroughly investigate the computational complexity of this problem and to show that it remains hard even in very restricted cases and for many parameterizations. We further put the results into context with an existing algorithm, essentially showing that this algorithm cannot be significantly improved.

Chapter 4: Temporal Separators. Coming from the investigation of temporal paths, we continue to investigate the problem of destroying all paths between two dedicated vertices by removing vertices from the graph. This problem falls into the "graph modification" category. Finding these so-called *separators* is a classic graph problem which can be solved efficiently and which also has been transferred to the temporal setting fairly early.

Our contribution here is twofold: Firstly, we investigate the computational complexity on a quite restricted class of temporal graphs, which we call *temporal unit interval graphs*, a generalization of unit interval graphs[1] to the temporal setting. We show that, in general, finding temporal separators remains computationally hard on temporal unit interval graphs but we identify an additional restriction which makes the problem solvable in polynomial time. We use this to develop a fixed-parameter algorithm for a parameter that measures the "distance" to the easy case. Secondly, we investigate the computational complexity of finding separators that destroy all *restless* temporal paths, which we discuss in Chapter 3. Our main result is that the problem of finding these *restless* temporal separators is hard for a complexity class located in the second level of the polynomial time hierarchy.

Chapter 5: Temporal Matchings. In this chapter we introduce a model for matchings in temporal graphs. Finding matchings is, again, a very fundamental task in graph algorithmics that receives much attention in the scientific community to this day. The model of temporal matching we consider informally works as follows. In a temporal graph, if two vertices are matched at some point in time, then they have to "recharge" afterwards, meaning that they cannot be matched again for a certain number of time steps.

We investigate the computational complexity of this problem and show that it is hard even in very restricted instances. To show our main result, we employ a temporal analogue of line graphs[2] and obtain some results that may be of independent interest. We further discuss how our results imply that known algorithms for this problem presumably cannot be improved significantly.

Chapter 6: Temporal Coloring. In this chapter we lift the classic graph coloring problem to the temporal setting. We introduce a model where every edge has to be colored properly (that is, the endpoints are colored differently) at least once in every time interval of a certain length. This problem does not really fit into any of the two categories described in the previous section but can rather be seen as the problem of deciding whether a given temporal graph has a certain property.

We show that this problem is computationally hard in very restricted cases, even if we only have two colors. Note that the classic analogue, that is, checking bipartiteness, can be solved efficiently. We present rather simple algorithms to solve

[1] A graph is a *unit interval graph* if its vertices can be assigned to intervals of the rational numbers of unit length, such that two vertices are connected by an edge if and only if the two corresponding intervals overlap.

[2] A *line graph* of a graph has a vertex for each edge of the original graph, and two vertices are connected if the corresponding edges share an endpoint.

this problem, however, we consider the main contribution that we show that these algorithms presumably cannot be improved significantly.

Chapter 7: Temporal Cliques and *s*-Plexes. In this chapter we investigate finding cliques and *s*-plexes[3] in temporal graphs. Finding and enumerating cliques is a classic problem that is very well investigated. This problem clearly falls into the "substructure detection" category. We propose a model for temporal *s*-plexes that is a canonical generalization of an existing model for temporal cliques. Here, a temporal *s*-plex is a set of vertices together with a time interval, such that in each time window of a certain size within this time interval, we have that every vertex is connected to at least $s-1$ of the other vertices in the temporal *s*-plex. A temporal clique is simply a temporal 1-plex.

Since finding cliques in graphs is computationally hard, this generalization clearly is hard as well. Our main contribution is a fixed-parameter algorithm that enumerates all temporal *s*-plexes in a given temporal graph, where we use a temporal analogue of degeneracy[4] (a measure of sparsity) as a parameter.

Chapter 8: Temporal Cluster Editing. In this chapter we analyze the computational complexity of clustering a temporal graph. The classic problem we lift to the temporal setting here is the problem of modifying the edges of a graph such that every connected component becomes a clique. Hence, here we have a problem that clearly falls into the "graph modification" category. In our model, we want to modify every layer, that is, the graph consisting of all edges that are present at a certain time step, such that every of its connected components is a clique. However, we also want that these connected components look somewhat similar at every time step.

The classic version of this problem is computationally hard and hence the generalization to the temporal setting is as well. Our main contribution is a fixed-parameter algorithm for the parameter "number of edge modifications" plus the measure of similarity of the resulting clusterings. We further show that there is an efficient preprocessing procedure that can provably reduce the size of the input instance to be independent of the number of vertices of the original input instance.

Short Summary. The thesis contains six chapters each exploring a temporal version of a classic graph problem. It starts with an introduction and a chapter introducing all necessary definitions and notations and concludes with a chapter discussing

[3] An *s-plex* is a set of vertices where every vertex is connected to all but $s-1$ of the other vertices in the set.

[4] The *degeneracy* of a graph is the smallest integer d such that each subgraph contains a vertex with degree at most d.

general take home messages and future work directions. Moreover, each of the six chapters dedicated to a specific problem also provides concluding remarks and future research directions tailored to the problem under investigation.

CHAPTER 2

Preliminaries and Notation

In this chapter, we introduce all concepts, notations, and terminology that are used in and relevant for many chapters of this thesis. Each chapter also contains a "preliminaries" section introducing concepts and notation specific to that chapter.

We start with some basic mathematical definitions. We use $\mathbb{N}$ to denote the natural numbers without zero. We refer to an interval as a contiguous ordered set of natural numbers. Formally, an *interval* is an ordered set

$$I = [a,b] := \{n \mid n \in \mathbb{N} \wedge a \leq n \leq b\},$$

where $a, b \in \mathbb{N}$. Further, let $[a] := [1,a]$. Given a set S, we denote the set of all subsets of size two by $\binom{S}{2}$, that is, $\binom{S}{2} := \{\{a,b\} \mid a \in S \wedge b \in S \wedge a \neq b\}$.

2.1 Static Graphs

We use standard notation from (static) graph theory [Die16]. Unless stated otherwise, we assume graphs in this thesis to be *undirected* and *simple*. To clearly distinguish them from temporal graphs, they are sometimes referred to as *static* graphs. Given a (static) graph $G = (V,E)$ with $E \subseteq \binom{V}{2}$, we denote by $V(G) := V$ and $E(G) := E$ the sets of its vertices and edges, respectively. We call two vertices $u, v \in V$ *adjacent* if $\{u,v\} \in E$ and we call u and v the *endpoints* of edge $\{u,v\}$. Two edges $e_1, e_2 \in E$ are *adjacent* if $e_1 \cap e_2 \neq \emptyset$. The *neighborhood* of a vertex $v \in V$ is the set $N_G(v) = \{u \mid \{u,v\} \in E\}$. We call the size of the neighborhood of a vertex its *degree*. If a vertex has degree zero, then we say that this vertex is *isolated*. For some vertex subset $V' \subseteq V$, we denote by $G[V']$ the *induced* subgraph of G on the vertex set V', that is, $G[V'] = (V', E')$ where $E' = \{\{v,w\} \mid \{v,w\} \in E \wedge v \in V' \wedge w \in V'\}$. By P_n we denote a graph that is a path with n vertices.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a bijection $\sigma : V_1 \to V_2$ such that for all $u, v \in V_1$ we have that $\{u,v\} \in E_1$ if and only if $\{\sigma(u), \sigma(v)\} \in E_2$. Given a graph $G = (V,E)$ and an edge $\{u,v\} \in E$, *subdividing* the edge $\{u,v\}$ results in a graph isomorphic to $G' = (V', E')$ with $V' = V \cup \{w\}$ for some $w \notin V$ and $E' = (E \setminus \{\{u,v\}\}) \cup \{\{v,w\},\{u,w\}\}$. We call a graph H a *subdivision* of a graph G if there is a sequence of graphs $G_1, G_2, \ldots, G_x$ with $G_1 = G$ such that for each $G_i = (V_i, E_i)$ with $i < x$ there is an edge $e \in E_i$ and subdividing e results in a graph isomorphic to G_{i+1},

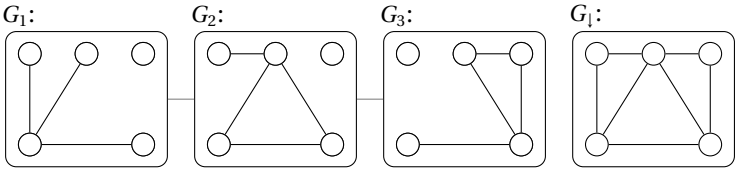


Figure 2.1: Example of a temporal graph with lifetime three and its underlying graph.

and G_x is isomorphic to H. We call H a *topological minor* of G if there is a subgraph G' of G that is a subdivision of H. We call H an *induced topological minor* of G if there is an *induced* subgraph G' of G that is a subdivision of H.

2.2 Temporal Graphs

Since temporal graph theory and temporal graph algorithmics are still rather young research fields, notation and terminology is not yet completely standardized and in the literature one can find many synonyms for terminology or mathematically equivalent definitions that use different notation [Boc+14, Cas+12, CF13a, CF13b, Hol15, HS12, HS13, HS19, LVM18, Mic16]. In the following we introduce the notation and terminology we use in this thesis.

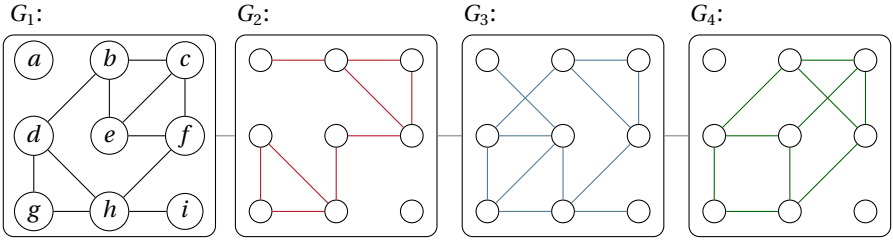
Temporal graphs are the most important mathematical objects in this thesis. They are graphs where the edge set may change over a discrete set of time steps. We model a temporal graph as a vertex set together with an ordered list of edge sets over this vertex set.

Definition 2.1 (Temporal Graph)**.** An (undirected, simple) *temporal graph* is a tuple $\mathcal{G} = (V, E_1, E_2, \dots, E_\ell)$ (or $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ for short), with $E_i \subseteq \binom{V}{2}$ for all $i \in [\ell]$.

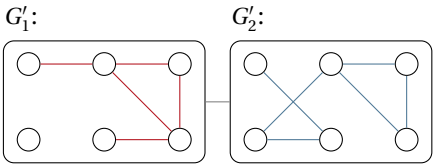
We call $\ell(\mathcal{G}) := \ell$ the *lifetime* of $\mathcal{G}$. As with static graphs, we assume all temporal graphs in this thesis to be undirected and simple.

We call the graph $G_i(\mathcal{G}) = (V, E_i(\mathcal{G}))$ the *layer* i of $\mathcal{G}$ where $E_i(\mathcal{G}) := E_i$. If $E_i = \emptyset$, then G_i is a *trivial* layer. We call layers G_i and G_{i+1} *adjacent*. We call i a *time step*. If an edge e appears at time i, that is, $e \in E_i$, then we say that e has *time stamp* i. We further denote $V(\mathcal{G}) := V$. The *underlying graph* $G_\downarrow(\mathcal{G})$ of $\mathcal{G}$ is defined as $G_\downarrow(\mathcal{G}) := (V, \bigcup_{i=1}^{\ell(\mathcal{G})} E_i(\mathcal{G}))$. To improve readability, we remove $(\mathcal{G})$ from the introduced notations whenever it is clear from the context. For an example of a temporal graph and its underlying graph see Figure 2.1.

For every $v \in V$ and every time step $t \in [\ell]$, we denote the *appearance of vertex v at time t* by the pair (v, t). For every $t \in [\ell]$ and every $e \in E_t$ we call the pair (e, t) a *time*



(a) A temporal graph $\mathcal{G}$.



(b) Induced temporal subgraph $\mathcal{G}' = \mathcal{G}[X]|_{[2,3]}$ with $X = \{a, b, c, e, d, f\}$.

Figure 2.2: Example of a temporal graph and an induced temporal subgraph.

edge. We assume that the *size* (for example when referring to input sizes in running time analyzes) of $\mathcal{G}$ is $|\mathcal{G}| := |V| + \sum_{i=1}^{\ell} |E_i|$, that is, we do not assume that we have compact representations of temporal graphs.

Induced Temporal Subgraphs. We use a canonical analogue to induced subgraphs of static graphs in the temporal setting. For a vertex subset $X \subseteq V$ and a time interval $[a, b] \subseteq [\ell]$, we define the *induced temporal subgraph* of $\mathcal{G}$ by X and $[a, b]$ as $\mathcal{G}[X]|_{[a,b]} := (X, (E'_i)_{i \in [\ell']})$ where $\ell' = b - a + 1$ and $E'_i = \{\{v, w\} \mid v \in X \wedge w \in X \wedge \{v, w\} \in E_{i+a-1}\}$ for $i \in [\ell']$. For an illustration see Figure 2.2. If $X = V$, then we write $\mathcal{G}|_{[a,b]}$ as short form for $\mathcal{G}[X]|_{[a,b]}$, and if $[a, b] = [\ell]$, then we write $\mathcal{G}[X]$ as short form for $\mathcal{G}[X]|_{[a,b]}$. Furthermore, we define $\mathcal{G} - X := \mathcal{G}[V \setminus X]$.

Time Windows. We now define Δ-time windows, which we make use of in several different problem settings we investigate in this thesis. Let $\Delta \leq \ell$. For every time step $t \in [\ell - \Delta + 1]$, the Δ-*window* $W_t^\Delta = [t, t + \Delta - 1]$ is the sequence of the Δ consecutive time steps $t, t + 1, \ldots, t + \Delta - 1$. If $\Delta < 1$, then we define the Δ-window for any time step to be the empty set. Furthermore, we denote by $E_{W_t^\Delta} = \bigcup_{i \in W_t^\Delta} E_i$ the union of all edges appearing at least once in the Δ-time window W_t^Δ. We remark that in the literature, the symbol Δ is also often used to denote the maximum degree of a graph.

In the context of this thesis, however, we exclusively use this symbol in the context of time windows.

2.3 Parameterized Complexity

We use standard notation and terminology from parameterized complexity theory [Cyg+15, DF13, DF99, FG06, Fom+19, Nie06] and give here a brief overview of the most important concepts that are used in this thesis. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. We call the second component the *parameter* of the problem. A parameterized problem is *fixed-parameter tractable* (in the complexity class FPT) if there is an algorithm that solves each instance (I, r) in $f(r) \cdot |I|^{O(1)}$ time, for some computable function f. A decidable parameterized problem L admits a *polynomial kernel* if there is a polynomial-time algorithm that transforms each instance (I, r) into an instance (I', r') such that $(I, r) \in L$ if and only if $(I', r') \in L$ and $|(I', r')| \in r^{O(1)}$. If a parameterized problem is hard for the parameterized complexity class W[1] or W[2], then it is (presumably) not in FPT. Informally, parameterized problems that are hard for W[1] are at least as hard as CLIQUE[5] parameterized by the solution size, and parameterized problems that are hard for W[2] are at least as hard as HITTING SET[6] parameterized by the solution size. The complexity classes W[1] and W[2] are closed under parameterized reductions, which may run in FPT-time and additionally set the new parameter to a value that exclusively depends on the old parameter. If a parameterized problem is NP-hard for constant parameter values, then the problem is para-NP-hard.

Kernelization Lower Bounds. We employ the cross-composition framework [BJK14, Bod+09, Dru15, Fom+19, FS11] to refute the existence of a polynomial kernel for a parameterized problem under the assumption that NP $\not\subseteq$ coNP/poly, the negation of which would cause a collapse of the polynomial-time hierarchy to the third level. Informally, in a cross-composition, we have to *compose* many problem instances of an NP-hard problem into one big instance of the problem we want to investigate. This composition should then have the property that the big instance is either a YES-instance if and only if at least one of the input instances is a YES-instance (in the case of OR-cross-compositions), or if and only if all input instance are YES-instances (in the case of AND-cross-compositions). This then refutes polynomial kernels for

[5] In CLIQUE we are given a graph G and a "solution size" k, and are asked to decide whether G contains a set of k vertices that are all pairwise connected by an edge.

[6] In HITTING SET we are given a universe U, a collection of subsets of U, and a "solution size" k, and are asked to decide whether it is possible to select k elements of U such that every subset in the collection contains at least one selected element.

the problem under investigation when parameterized by any parameter that only depend on the maximum size of the input instances (and not on the number of input instances). In order to formally introduce the framework, we need some definitions first.

An equivalence relation R on the instances of some problem L is a *polynomial equivalence relation* if

1. one can decide for each two instances in time polynomial in their sizes whether they belong to the same equivalence class, and
2. for each finite set S of instances, R partitions the set into at most $(\max_{x \in S} |x|)^{O(1)}$ equivalence classes.

Using this, we can now define OR-cross-compositions and AND-cross-compositions.

An *OR-cross-composition* of a problem $L \subseteq \Sigma^*$ into a parameterized problem P (with respect to a polynomial equivalence relation R on the instances of L) is an algorithm that takes n R-equivalent instances $x_1, \ldots, x_n$ of L and constructs in time polynomial in $\sum_{i=1}^{n} |x_i|$ an instance (x, k) of P such that

1. k is polynomially upper-bounded in $\max_{1 \leq i \leq n} |x_i| + \log(n)$ and
2. (x, k) is a YES-instance of P if and only if there is an $i \in [n]$ such that x_i is a YES-instance of L.

If an NP-hard problem L OR-cross-composes into a parameterized problem P, then P does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly [BJK14, Bod+09, FS11].

AND-cross-compositions are defined analogously. An *AND-cross-composition* of a problem $L \subseteq \Sigma^*$ into a parameterized problem P (with respect to a polynomial equivalence relation R on the instances of L) is an algorithm that takes n R-equivalent instances $x_1, \ldots, x_n$ of L and constructs in time polynomial in $\sum_{i=1}^{n} |x_i|$ an instance (x, k) of P such that

1. k is polynomially upper-bounded in $\max_{1 \leq i \leq n} |x_i| + \log(n)$ and
2. (x, k) is a YES-instance of P if and only if x_i is a YES-instance of L for every $i \in [n]$.

If an NP-hard problem L AND-cross-composes into a parameterized problem P, then P does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly [BJK14, Dru15].

Exponential Time Hypothesis. The Exponential Time Hypothesis (ETH) implies that there is no algorithm for 3-SAT that has a running time in $2^{o(n+m)}$, where n and m denote the number of variables and clauses, respectively, of a 3-SAT formula [IP01, IPZ01]. The ETH is typically used to obtain fine-grained running time lower bounds for exponential-time algorithms. We remark that all NP-hardness reductions also yield some running time lower bounds assuming the ETH. However, in this thesis, we only discuss ETH-based running time lower bounds if they show that the running time of some existing algorithm, or some algorithm that we present in this thesis, presumably cannot be improved. If this is not the case, then we omit discussing which ETH-based running time lower bounds the reductions presented in this thesis imply.

2.4 (Temporal) Graph Parameters

In parameterized complexity, we not only want to investigate a problem parameterized by its sometimes called "natural problem parameters" such as for example the solution size, but also with parameters that quantify some structural properties of the input (temporal) graph. We call parameters of this type *structural graph parameters*. More formally, we call a graph parameter *structural* if there is a function mapping graphs to natural numbers that describes the parameter value of a given (temporal) graph.

Static Graph Parameters. In the following, we present definitions of the (static) structural graph parameters we consider in this thesis.

Feedback Vertex Number: The *feedback vertex number* of a static graph $G = (V, E)$ is the cardinality of a minimum vertex subset $V' \subseteq V$ such that $G - V'$ is a forest.

Degeneracy: The *degeneracy* of a static graph G is the smallest integer $d \in \mathbb{N}$ such that each subgraph G' of G contains a vertex v with degree at most d.

Domination Number: The *domination number* of a static graph $G = (V, E)$ is the cardinality of a minimum vertex subset $V' \subseteq V$ such that every vertex $v \in V$ is either contained in V' or has a neighbor that is contained in V'.

Maximum Degree: The *maximum degree* of a static graph $G = (V, E)$ is $\max_{v \in V} |N(v)|$.

Treedepth: The *treedepth* of a static graph $G = (V, E)$ is the minimum height of a rooted forest $F = (V, E')$ with the property that every edge of G connects a pair of nodes that have an ancestor-descendant relationship to each other in F.

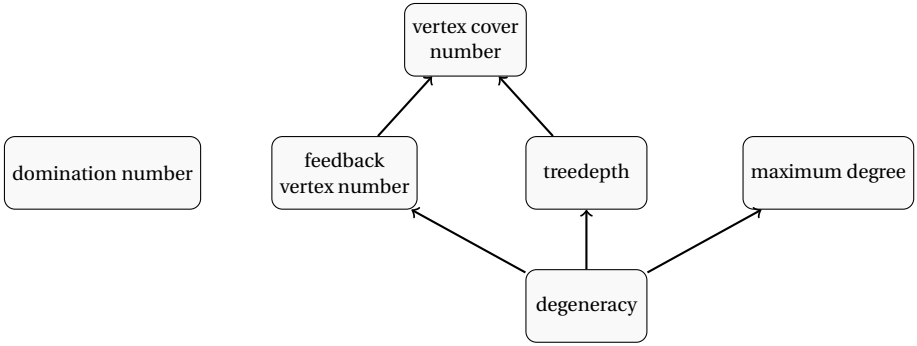


Figure 2.3: A Hasse diagram of the partial ordering for the structural parameters used in this thesis. An arrow from a parameter p to another parameter p' indicates that p' is larger than p. Parameter pairs that are not connected by an arrow are known to be incomparable. For details we refer to Sorge and Weller [SW19].

Vertex Cover Number: The *vertex cover number* of a static graph $G = (V, E)$ is the cardinality of a minimum vertex subset $V' \subseteq V$ such that every edge $e \in E$ has at least one endpoint in V'.

Note that most of the mentioned parameters also have equivalent alternative definitions. Further note that parameters such as "number of vertices" or "number of edges" could also be regarded as structural graph parameters.

Comparison of Parameters. In the following we describe a partial ordering of structural graph parameters. Let p_1 and p_2 be two structural graph parameters, then we say that p_1 is *larger* than p_2 if there is a function f such that for all graphs G we have that $p_2(G) \leq f(p_1(G))$. In other words, a parameter p_1 is larger than a parameter p_2 if p_1 can be used to upper-bound p_2. Furthermore, if p_1 is larger than p_2, then we say that p_2 is *smaller* than p_1. This notion creates a partial ordering on structural graph parameters which we visualize for the parameters introduced above in Figure 2.3. For a larger view on this partial ordering for structural graph parameters we refer to Sorge and Weller [SW19]. If for two parameters p and p' have the property that p is neither larger nor smaller than p', then we say that p and p' are *incomparable*.

Note that in the context of parameterized complexity theory, this partial ordering allows us to draw conclusions of the following kind. Let p and p' be two structural graph parameters such that p' is larger than p. If a problem is fixed-parameter tractable with respect to a parameter p, then it is also fixed-parameter tractable

with respect to the larger parameter p'. If a problem is hard for a parameterized complexity class (such as for example W[1]) with respect to p', then it is also hard for this complexity class for the smaller parameter p.

Temporal Graph Parameters. There are several canonical ways to transfer a structural graph parameter from the static to the temporal setting. Let p be a structural graph parameter. Two of the most obvious ways are the following. Let $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ be a temporal graph.

1. We can consider the maximum of the parameter values of all layers, that is, $\max_{i\in[\ell]} p(G_i)$, which we abbreviate with $p_{\max}$.
2. We can consider the parameter value of the underlying graph, that is, $p(G_\downarrow)$, which we abbreviate with $p_\downarrow$.

When comparing these two options, we can observe that we have different behaviors depending on p. It is clear that any layer of a temporal graph is a (non-induced) subgraph of the underlying graph, that is, we can obtain each layer of a temporal graph by removing edges from the underlying graph. Hence we have the following situation: Let $G' = (V, E')$ be a subgraph of $G = (V, E)$, that is, $E' \subseteq E$. If we have that $p(G) \geq p(G')$, then it follows that $p_\downarrow$ is a larger parameter than $p_{\max}$. This is the case for most of the parameters we consider in this thesis, such as vertex cover number, feedback vertex number, treedepth, maximum degree, and degeneracy. However, there are also parameters where we have the opposite relation between $p_{\max}$ and $p_\downarrow$. One parameter with this property that we consider is the domination number. It is easy to see that this parameter is non-decreasing under edge removal. Consider the extreme cases of a complete graph (which has domination number one) and an edgeless graph (where the domination number equals the number of vertices). Note that there are also cases where $p_{\max}$ and $p_\downarrow$ are incomparable.

Of course there are also many temporal graph parameters that do not fall into this scheme. In Chapters 4 and 7 we will see examples of parameters of this kind.

2.5 Temporal Graph Problems vs. Multi-Layer Graph Problems

Multi-layer graph problems are related to temporal graph problems because the mathematical object (a multi-layer graph) is very similar to a temporal graph. A *multi-layer graph* is a set of vertices together with a collection of edge sets [Kiv+14]. In other words, if we remove the natural ordering of the edge sets from a temporal graph, we obtain a multi-layer graph.

From a motivation and modelling standpoint, the multi-layer graphs are used if the different layers model incomparable or orthogonal properties of the data. For

example, different layers in a multi-layer graph modelling a social network can refer to different types of relations (friends, family, co-workers, etc.) or different social network platforms.

However, as we will see in this thesis, we can also have the case that in a temporal graph problem the ordering of the layers does not play an important role. Formally, we say that a temporal graph problem may be treated as a *multi-layer graph problem* if it has the following property. Given a problem instance with a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$, it holds that for every permutation $\pi : [\ell] \to [\ell]$ we have that the instance containing $\mathcal{G}$ is a YES-instance if and only if the instance where $\mathcal{G}$ is replaced by $\mathcal{G}' = (V, (E_{\pi(i)})_{i \in [\ell]})$ is a YES-instance.

CHAPTER 3

Restless Temporal Paths

The first problem we investigate in this thesis is a temporal version of one of the most fundamental primitives in graph algorithmics: Computing a (shortest) path between two vertices in a static graph. The static problem has been studied for decades and also in the temporal setting it has received substantial attention in recent years [Wu+16, XFJ03]. In a nutshell, temporal paths have to respect time, that is, they may only move forward in time. More formally, the time edges used by a temporal path either need to have increasing or at least non-decreasing time stamps. It is well-known that computing temporal paths can be done in polynomial time. We study a natural variant, where temporal paths may only dwell a certain given amount of time steps in any vertex, which we call *restless temporal paths*. This small modification creates a significant change in the computational complexity of the task of finding temporal paths. We show that finding restless temporal paths is NP-complete and give a thorough analysis of the (parameterized) computational complexity of this problem. In particular, we show that the problem remains computationally hard on temporal graphs with three layers and is W[1]-hard when parameterized by the feedback vertex number of the underlying graph.

This chapter is based on the paper "The computational complexity of finding temporal paths under waiting time constraints" by Casteigts et al. [Cas+20].

3.1 Introduction

Checking connectivity and computing shortest paths between vertices in a graph is one of the most important tasks in graph algorithmics, both as a direct problem setting and as a subroutine for numerous other applications. On static graphs, these problems have been studied for decades. The study of temporal paths (sometimes also called "journeys") and temporal connectivity gained more and more attention in the recent years. Intuitively, a temporal path is a path through a temporal graph that "respects" time, that is, the time edges it uses have increasing or non-decreasing time stamps. Even with this informal description we can already see that temporal paths have some properties that usual paths in static (undirected) graphs do not have.

- Temporal paths are "directed" even in *undirected* temporal graphs: Since the time stamps of the edges used by the path have to increase (or may not decrease), it is not necessarily possible to traverse a temporal path in the opposite direction. This implies that the relation of "being connected" between vertices is not symmetric in a temporal graph.

- Temporal paths cannot necessarily be "composed": If there is a temporal path from a vertex v to a vertex w and a temporal path from vertex w to a vertex u, there is not necessarily a temporal path from v to u, since the path from w to u might be "too early". This implies that in temporal graphs the relation of "being connected" between vertices is not transitive.

This puts some challenges on designing algorithms to check connectivity that are not present in the static setting. We are in a somewhat similar situation when we want to compute "shortest" temporal paths. There are several canonical notions of a "shortest" or "optimal" temporal path. We informally describe the three most important ones in the following.

- A *shortest* temporal path is a temporal path between two vertices that uses a minimum number of time edges.

- A *fastest* temporal path is a temporal path between two vertices with a minimum difference between the time stamps of the first and last time edge used by the path.

- A *foremost* temporal path is a temporal path between two vertices with a minimum time stamp on its last time edge.

Each one of these optimal temporal paths has some properties that shortest paths in static graphs do not have, in particular, a "subpath" of an optimal path is not necessarily optimal. Furthermore, we have that also temporal walks, which may visit the same vertex multiple times, can be optimal (at least in the case of "fastest" and "foremost"). All what we have seen above makes computing optimal temporal paths, while still being polynomial-time solvable for the above three mentioned optimality criteria, more challenging than computing their static counterpart.

In this chapter, we consider yet another natural type of temporal paths, which we call *restless temporal paths*. Intuitively, a restless temporal path cannot stay in any vertex longer than some given time period Δ. Consider the following motivating example. In disease spreading scenarios it is very natural to model the spreading process with a temporal network, where vertices correspond to individuals and time

edges to for example physical contact, where infections can happen [Hol16]. Many diseases have the property that infected individuals can recover (or die) after some fixed time period and are immune afterwards[7], the disease travels along restless paths: It has to infect another individual before its current host recovers. If we assume that a recovered person is immune to the disease afterwards, then the disease cannot infect a host twice. Hence, it has to travel along a path with bounded waiting times. This model for temporal paths has not received much attention of late. It has been considered as a natural variant for temporal paths [HS12, PS11], but only recently this model was studied from a computational perspective [Him+19, Him18, Zsc17] where, however, mainly restless temporal walks were considered. Restless temporal walks, that is, temporal paths with restricted maximum waiting times that may visit vertices multiple times, can be found in polynomial time [Him+19]. In this chapter, we focus on restless temporal paths, that is, any vertex can be visited at most once. Surprisingly and in stark contrast to both restless temporal walks and non-restless temporal paths, we show that it is NP-complete (even in very restricted cases) to decide whether there exists a restless temporal path between two vertices.

3.1.1 Related Work

To the best of our knowledge, temporal paths and walks were first considered by Göbel, Cerdeira, and Veldman [GCV91] in the context of information flow over time. Other early work on temporal paths includes Berman [Ber96], where they were also considered in the context of network flows, and the work of Kempe, Kleinberg, and Kumar [KKK02], where they were considered in the context of connectivity and separation problems. More systematic work with a focus on computing (optimal) temporal paths as an algorithmic primitive (as opposed to a subroutine) was conducted for example by Xuan, Ferreira, and Jarry [XFJ03], Wu et al. [Wu+16], and Himmel et al. [Him+19]. They gave polynomial-time algorithms for finding optimal temporal paths for several optimality criteria. Casteigts et al. [Cas+15b] studied the problem from a distributed computing perspective.

Maximum waiting times have been considered as a natural variant for temporal paths [HS12, PS11], however only recently this model has been studied from an algorithmic standpoint [Him+19] and only in the context of temporal walks, where in particular, they showed that restless temporal walks can be computed in polynomial time. Casteigts et al. [Cas+15a] studied temporal paths with maximum waiting times in the context of analyzing the expressivity of temporal graphs. In the context of

[7] This is a standard assumption in the *SIR-model* (Susceptible-Infected-Recovered), a canonical spreading model for diseases that give immunity upon recovery [Bar16, New18].

temporal flows, a concept somewhat similar to maximum waiting times, called "vertex buffers", has been considered by Akrida et al. [Akr+19a].

Further work related to temporal paths and connectivity in temporal graphs includes research on temporal spanners and temporally connected graphs [AF16, Akr+17, CPS19, MMS19], temporal graph exploration [Akr+19b, AMS19, BZ19, EHK15, Erl+19, ES18, FMS13], modifying temporal graphs to increase or decrease connectivity [EMS21, Enr+19], and a temporal version of the TRAVELING SALES PERSON problem [MS16]. For related work on temporal separators we refer to Section 4.1.1.

3.1.2 Our Contributions and Organization of the Chapter

Our contributions are mainly computational hardness results. To the best of our knowledge, we are the first to establish NP-hardness of the problem of computing a restless temporal path between two vertices. We show that the problem is NP-hard even if the input temporal graph has only three layers. The reduction we present also implies some running time lower bounds based on the Exponential Time Hypothesis. We further show that finding restless temporal paths is W[1]-hard when parameterized by the feedback vertex number of the underlying graph of the input temporal graph. We discuss how these computational hardness results relate to algorithmic results from Casteigts et al. [Cas+20] that are not part of this thesis.

In Section 3.2 we formally introduce all necessary concepts related to temporal paths that we need in this chapter, we formally define our problem setting and report some basic observations about the problem. In Section 3.3 we present our main computational hardness results and discuss how they relate to known tractability results. We conclude in Section 3.4.

3.1.3 Further Contributions of the Manuscript this Chapter is Based on

Additionally to the contributions we present in this chapter, Casteigts et al. [Cas+20] show that finding shortest restless temporal paths is fixed-parameter tractable when parameterized by the maximum number of time edges used by the path. They further show that finding restless temporal paths is fixed-parameter tractable when parameterized by the feedback edge number of the underlying graph. Finally, they introduce a novel temporal version of the "feedback vertex number"-parameter which they call *timed feedback vertex number*. They show how to compute this parameter and that finding restless temporal paths is fixed-parameter tractable when parameterized by the timed feedback vertex number.

3.2 Preliminaries

In this section, we formally introduce the most important concepts related to temporal paths and walks and give the formal problem definitions of RESTLESS TEMPORAL (s,z)-PATH and SHORT RESTLESS TEMPORAL (s,z)-PATH.

3.2.1 Temporal Walks and Paths

Intuitively, a temporal path (sometimes also called "journey") is a path through a temporal graph that respects time, that is, the time edges it uses have non-decreasing time stamps. A temporal walk may visit the same vertex multiple times. Formally, these two concepts are defined as follows.

Definition 3.1 (Temporal Walk / Temporal Path)**.** A *temporal walk* of length n from vertex s to vertex z in a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ is a sequence $P = \big((\{s = v_0, v_1\}, t_1), (\{v_1, v_2\}, t_2), \ldots, (\{v_{n-1}, v_n = z\}, t_n)\big)$ of edges together with time stamps such that for all $i \in [n]$ we have that $\{v_{i-1}, v_i\} \in E_i$ and for all $i \in [n-1]$ we have that $t_i \leq t_{i+1}$. Moreover, we call P a *temporal path* of length n if $v_i \neq v_j$ for all $i, j \in \{0, \ldots, n\}$ with $i \neq j$.

Given a temporal path $P = \big((\{v_0, v_1\}, t_1), (\{v_1, v_2\}, t_2), \ldots, (\{v_{n-1}, v_n\}, t_n)\big)$, we denote the set of vertices visited by P by $V(P) = \{v_0, v_1, \ldots, v_n\}$.

3.2.2 Restless Temporal Walks and Paths

A *restless* temporal path is not allowed to wait an arbitrary amount of time in a vertex, but has to leave any vertex it visits within the next Δ-window, for some given value for Δ. Analogously to the non-restless case, a restless temporal walk may visit a vertex multiple times. Formally, they are defined as follows.

Definition 3.2 (Restless Temporal Walk / Restless Temporal Path)**.** A Δ*-restless temporal walk* of length n from vertex s to vertex z in a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ is a sequence $P = \big((\{s = v_0, v_1\}, t_1), (\{v_1, v_2\}, t_2), \ldots, (\{v_{n-1}, v_n = z\}, t_n)\big)$ of edges together with time stamps such that for all $i \in [n]$ we have that $\{v_{i-1}, v_i\} \in E_i$ and for all $i \in [n-1]$ we have that $t_i \leq t_{i+1} \leq t_i + \Delta$. Moreover, we call P a Δ*-restless temporal path* of length n if $v_i \neq v_j$ for all $i, j \in \{0, \ldots, n\}$ with $i \neq j$. We say that P *respects* the maximum waiting time Δ.

In Figure 3.1 we give an example: we are given the depicted temporal graph, vertices s and z, and the time bound $\Delta = 2$. Here, $((\{s,d\},2),(\{d,b\},4),(\{b,z\},6))$ is a Δ-restless temporal (s,z)-path, but $((\{s,b\},1),(\{b,z\},6))$ is not because the waiting time at b exceeds Δ. Furthermore $((\{s,b\},1),(\{b,c\},2),(\{c,d\},4),(\{d,b\},4),(\{b,z\},6))$

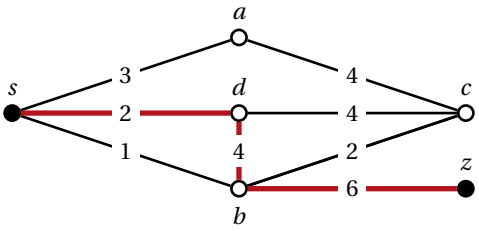


Figure 3.1: Example of a temporal graph whose edges are labeled with time stamps. Bold red edges form a 2-restless temporal (s,z)-path.

is a Δ-restless temporal *walk* but not a path because it visits vertex b twice. Finally, $((\{s,a\},3),(\{a,c\},4),(\{c,d\},4),(\{d,b\},4),(\{b,z\},6))$ is a Δ-restless temporal (s,z)-path for $\Delta = 2$ because the waiting time at the source is not taken into consideration.

Having Definition 3.2 definition at hand, we are ready to define the main decision problem of this chapter.

RESTLESS TEMPORAL (s,z)-PATH	
Input:	A temporal graph $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$, two distinct vertices $s,z \in V$, and an integer $\Delta \leq \ell$.
Question:	Is there a Δ-restless temporal path from s to z in $\mathcal{G}$?

We also consider a variant, where we want to find Δ-restless paths of a certain maximum length.

SHORT RESTLESS TEMPORAL (s,z)-PATH	
Input:	A temporal graph $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$, two distinct vertices $s,z \in V$, and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.
Question:	Is there a Δ-restless temporal path of length at most k from s to z in $\mathcal{G}$?

RESTLESS TEMPORAL (s,z)-PATH is the special case of SHORT RESTLESS TEMPORAL (s,z)-PATH for $k = |V|-1$, since no Δ-restless temporal path in a graph $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$ can have length more than $|V|-1$. Furthermore, both RESTLESS TEMPORAL (s,z)-PATH and SHORT RESTLESS TEMPORAL (s,z)-PATH are clearly contained in NP since, given a temporal path, we can easily verify in polynomial time whether it is Δ-restless.

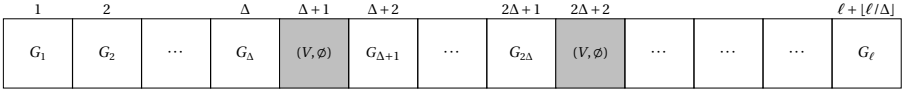


Figure 3.2: Inserting trivial layers to reduce RESTLESS TEMPORAL (s,z)-PATH on instances $(\mathcal{G},s,z,\Delta)$ to RESTLESS TEMPORAL (s,z)-PATH on instances $(\mathcal{G},s,z,\Delta+1)$.

3.2.3 Basic Observations

It is easy to observe that computational hardness of RESTLESS TEMPORAL (s,z)-PATH for some fixed value of Δ implies hardness for all larger values of Δ. This allows us to construct hardness reductions for small fixed values of Δ and still obtain general hardness results.

Observation 3.1. *For every fixed* Δ, RESTLESS TEMPORAL (s,z)-PATH *on instances* $(\mathcal{G},s,z,\Delta+1)$ *is computationally at least as hard as* RESTLESS TEMPORAL (s,z)-PATH *on instances* $(\mathcal{G},s,z,\Delta)$.

Proof. The result immediately follows from the observation that a temporal graph $\mathcal{G}$ contains a Δ-restless temporal (s,z)-path if and only if the temporal graph $\mathcal{G}'$ contains a $(\Delta+1)$-restless temporal (s,z)-path, where $\mathcal{G}'$ is obtained from $\mathcal{G}$ by inserting one trivial (that is, edgeless) layer after every Δ consecutive layers (see Figure 3.2). □

Furthermore, we can show that RESTLESS TEMPORAL (s,z)-PATH does not admit a polynomial kernel when parameterized by the number $|V|$ of vertices.

Proposition 3.2. RESTLESS TEMPORAL (s,z)-PATH *parameterized by the number* $|V|$ *of vertices does not admit a polynomial kernel for all* $\Delta \geq 1$ *unless NP* $\subseteq$ *coNP/poly.*

Proof. We provide an OR-cross-composition (for a definition see Section 2.3) from RESTLESS TEMPORAL (s,z)-PATH onto itself. Intuitively, we can just string together instances in the time axis such that the large instance contains a Δ-restless temporal (s,z)-path if and only if one of the original instances contains one.

We define an equivalence relation R as follows: Two instances $(\mathcal{G}=(V,(E_i)_{i\in[\ell]}),s,z,\Delta)$ and $(\mathcal{G}'=(V',(E'_i)_{i\in[\ell']}),s',z',\Delta')$ are equivalent under R if and only if $|V|=|V'|$ and $\Delta=\Delta'$. Clearly, R is a polynomial equivalence relation.

Now let $(\mathcal{G}_1=(V_1,(E_{1,i})_{i\in[\ell_1]}),s_1,z_1,\Delta_1),\ldots,(\mathcal{G}_n=(V_n,(E_{n,i})_{i\in[\ell_n]}),s_n,z_n,\Delta_n)$ be R-equivalent instances of RESTLESS TEMPORAL (s,z)-PATH. We construct a temporal graph $\mathcal{G}^\star=(V^\star,(E^\star_i)_{i\in[\ell^\star]})$ as follows. Let $|V^\star|=|V_1|$ and $s^\star,z^\star\in V^\star$. We identify all vertices s_i with $i\in[n]$ with each other and with $s^\star$, that is, $s^\star=s_1=\ldots=s_n$. Analogously, we identify all vertices z_i with $i\in[n]$ with each other and with $z^\star$, that is, $z^\star=z_1=$

$\ldots = z_n$. We arbitrarily identify the remaining vertices of the instances with the remaining vertices from $V^\star$, that is, let $V^\star \setminus \{s^\star, z^\star\} = V_1 \setminus \{s_1, z_1\} = \ldots = V_n \setminus \{s_n, z_n\}$. Now let $E^\star_1 = E_{1,1}, E^\star_2 = E_{1,2}, \ldots, E^\star_{\ell_1} = E_{1,\ell_1}$. Intuitively, the first instance $(\mathcal{G}_1 = (V_1, (E_{1,i})_{i\in[\ell_1]})$ essentially forms the first ℓ_1 layers of $\mathcal{G}^\star$. Then we introduce $\Delta_1 + 1$ trivial layers, that is, $E^\star_{\ell_1+1} = E^\star_{\ell_1+2} = \ldots = E^\star_{\ell_1+\Delta+1} = \emptyset$. Then we continue in the same fashion with the second instance and so on. We have that $\ell^\star = \sum_{i\in[n]} \ell_i + (n-1)\cdot(\Delta_1+1)$. Finally, we set $\Delta^\star = \Delta_1$.

This instance can be constructed in polynomial time and the number of vertices is the same as the vertices of the input instances, hence $|V^\star|$ is polynomially upper-bounded by the maximum size of an input instance. Furthermore, it is easy to check that $\mathcal{G}^\star$ contains a $\Delta^\star$-restless temporal $(s^\star, z^\star)$-path if and only if there is an $i \in [n]$ such that $\mathcal{G}_i$ contains a Δ_i-restless temporal (s_i, z_i)-path. This follows from the fact that all instances are separated in time by $\Delta_1 + 1$ trivial layers, hence no $\Delta^\star$-restless temporal $(s^\star, z^\star)$-path can use time edges from different original instances. Since Restless Temporal (s,z)-Path is NP-hard (Theorem 3.3) the result follows. □

3.2.4 Optimality Concepts for Temporal Paths

In contrast to the static setting, there are several canonical notions of a "shortest" or "optimal" temporal path. We informally describe the three most important ones in the following and then give their formal definitions. For convenience, we repeat the informal description from the introduction here.

- A *shortest* temporal path is a temporal path between two vertices that uses a minimum number of time edges.
- A *fastest* temporal path is a temporal path between two vertices with a minimum difference between the time stamps of the first and last time edge used by the path.
- A *foremost* temporal path is a temporal path between two vertices with a minimum time stamp on its last time edge.

Formally, they are defined as follows.

Definition 3.3 (Optimal Temporal Path)**.** Let $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ be a temporal graph with two vertices $s, z \in V$ and let $P = \big((\{s = v_0, v_1\}, t_1), (\{v_1, v_2\}, t_2), \ldots, (\{v_{n-1}, v_n = z\}, t_n)\big)$ be a temporal (s,z)-path. We call P

- a *shortest* temporal (s,z)-path, if there is no temporal (s,z)-path $P' = \big((\{s = v_0, v'_1\}, t'_1), (\{v'_1, v'_2\}, t'_2), \ldots, (\{v'_{n'-1}, v'_{n'} = z\}, t'_{n'})\big)$ with $n' < n$ in $\mathcal{G}$,

- a *fastest* temporal (s,z)-path, if there is no temporal (s,z)-path $P' = \big(\{s = v_0, v'_1\}, t'_1), (\{v'_1, v'_2\}, t'_2), \ldots, (\{v'_{n'-1}, v'_{n'} = z\}, t'_{n'})\big)$ with $t'_{n'} - t'_1 < t_n - t_1$ in $\mathcal{G}$, and
- a *foremost* temporal (s,z)-path, if there is no temporal (s,z)-path $P' = \big(\{s = v_0, v'_1\}, t'_1), (\{v'_1, v'_2\}, t'_2), \ldots, (\{v'_{n'-1}, v'_{n'} = z\}, t'_{n'})\big)$ with $t'_{n'} < t_n$ in $\mathcal{G}$.

These concepts transfer to Δ-restless temporal paths in a straightforward way. For an overview on further optimality concepts for temporal graphs we refer to Himmel et al. [Him+19] and Himmel [Him18].

3.2.5 Strict vs. Non-Strict Temporal Paths

Let us briefly discuss the concept of *strict* temporal paths and how it differs from the model we use here. In contrast to a (non-strict) temporal path (see Definition 3.1), the edges used by a strict temporal path need to have strictly increasing time stamps. Strict restless paths are defined in an analogous way. In this chapter, we focus on non-strict temporal paths and we believe that most of our results transfer to this setting, although we do not discuss this here. However, we remark that for some problem settings, the two models can behave quite differently [Zsc+20]. One obvious difference is that the length of a strict temporal path in a temporal graph is upper-bounded by the lifetime of the temporal graph, which allows for some tractability results for temporal graphs with small lifetime, whereas it is often not clear whether similar results also hold for the non-strict temporal path model [Zsc+20].

3.3 Finding Restless Temporal Paths

In this section we thoroughly analyze the computational hardness of Restless Temporal (s,z)-Path which of course transfers also to Short Restless Temporal (s,z)-Path.

3.3.1 NP-Hardness for Few Layers

We start by showing that Restless Temporal (s,z)-Path is NP-complete even if the lifetime of the input temporal graph is constant. To do this we present a reduction from Exact $(3,4)$-SAT [Tov84], a variant of 3-SAT where every variable appears in exactly four clauses and every clause has exactly three literals. The intuitive idea it that we create a variable gadget that a restless temporal path from s to z has to pass where the path can take two different routes for every variable, modeling an assignment for the formula. Then afterwards in the clause gadgets, the restless temporal path has to go through these vertices again at a later point in time, and this should only be possible if every clause is satisfied by the assignment modeled by the

first part of the path. The waiting time Δ prevents the restless temporal path from taking "shortcuts", that is, entering the clause gadgets earlier than it is supposed to. This is critical and the reason why this reduction fails for non-restless temporal paths.

Theorem 3.3. RESTLESS TEMPORAL (s,z)-PATH *is NP-complete for all* $\Delta \geq 1$ *and* $\ell \geq \Delta + 2$ *even if the underlying graph has maximum degree six and every edge has only one time stamp.*

Proof. We show this result by a polynomial-time reduction from the NP-complete EXACT (3,4)-SAT problem [Tov84]. The problem EXACT (3,4)-SAT asks whether a given Boolean formula ϕ is satisfiable and is in conjunctive normal form where each clause has exactly three literals and each variable appears in exactly four clauses.

Let ϕ be an instance of EXACT (3,4)-SAT with n variables and m clauses. We construct a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ with $\ell = 3$ (note that to get larger values for ℓ, we can simply append trivial layers to the constructed instance) consisting of a series of variable gadgets followed by a dedicated vertex s_n and then a series of clause gadgets. It is constructed in a way such that for $\Delta = 1$, any Δ-restless temporal (s,z)-path has to visit a vertex s_n and each possible Δ-restless temporal (s,s_n)-path represents exactly one variable assignment for the formula ϕ. Further we show that for any Δ-restless temporal (s,s_n)-path it holds that it can only be extended to a Δ-restless temporal (s,z)-path if and only if the Δ-restless temporal (s,s_n)-path represents a satisfying assignment for the formula ϕ.

Variable Gadget. We start by adding a vertex s to the vertex set V of $\mathcal{G}$. For each variable x_i with $i \in [n]$ of ϕ, we add nine fresh vertices to V: $x_i^{(1)}$, $x_i^{(2)}$, $x_i^{(3)}$, $x_i^{(4)}$, $\bar{x}_i^{(1)}$, $\bar{x}_i^{(2)}$, $\bar{x}_i^{(3)}$, $\bar{x}_i^{(4)}$, and s_i. Each variable x_i is represented by a gadget consisting of two disjoint path segments of four vertices each. One path segment is formed by $x_i^{(1)}$, $x_i^{(2)}$, $x_i^{(3)}$, and $x_i^{(4)}$ in that order and the second path segment is formed by $\bar{x}_i^{(1)}$, $\bar{x}_i^{(2)}$, $\bar{x}_i^{(3)}$, and $\bar{x}_i^{(4)}$ in that order. The connecting edges all appear exclusively at time step one, that is, $\{x_i^{(1)}, x_i^{(2)}\}$, $\{x_i^{(2)}, x_i^{(3)}\}$, and $\{x_i^{(3)}, x_i^{(4)}\}$ are added to E_1. Analogously for the edges connecting $\bar{x}_i^{(1)}$, $\bar{x}_i^{(2)}$, $\bar{x}_i^{(3)}$, and $\bar{x}_i^{(4)}$. Intuitively, if a Δ-restless temporal (s,z)-path passes the first segment, then this corresponds to setting the variable x_i to false. If it passes the second segment, then the variable is set to true. For all $i \in [n-1]$ we add the edges $\{x_i^{(4)}, s_i\}$, $\{\bar{x}_i^{(4)}, s_i\}$, $\{s_i, \bar{x}_{i+1}^{(1)}\}$, and $\{s_i, \bar{x}_{i+1}^{(1)}\}$ to E_1 and, additionally, we add $\{s, x_1^{(1)}\}$, $\{s, \bar{x}_1^{(1)}\}$, $\{x_n^{(4)}, s_n\}$, and $\{\bar{x}_n^{(4)}, s_n\}$ to E_1.

We can observe that there are exactly 2^n different temporal (s,s_n)-paths at time step one. Intuitively, each path represents exactly one variable assignment for the formula ϕ.

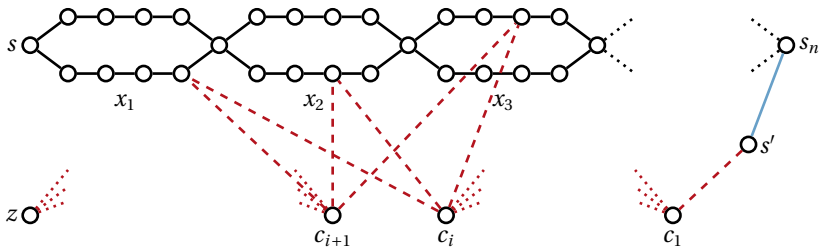


Figure 3.3: Illustration of the temporal graph constructed by the reduction in the proof of Theorem 3.3. An excerpt is shown with variable gadgets for x_1, x_2, and x_3 and the clause gadget for $c_i = (x_1 \vee x_2 \vee \neg x_3)$, where x_1 appears for the fourth time, x_2 appears for the third time, and x_3 also appears for the third time. Black edges appear at time step one, the light blue edge $\{s_n, s'\}$ appears at time step two, and the dashed red edges appear at time step three.

Clause Gadget. We add a vertex z to V. For each clause c_j with $j \in [m]$ we add a fresh vertex c_j to V. We further add a vertex s' to V and add the edge $\{s_n, s'\}$ to E_2. Let x_i (or $\bar{x}_i$) be a literal that appears in clause c_j and let this be the kth appearance of variable x_i in ϕ. Then, we add the edges $\{c_j, x_i^{(k)}\}, \{x_i^{(k)}, c_{j+1}\}$ (or $\{c_j, \bar{x}_i^{(k)}\}, \{\bar{x}_i^{(k)}, c_{j+1}\}$) to E_3 (where $c_{m+1} = z$). Finally, we add the edge $\{s', c_1\}$ to E_3.

Hence, there are exactly 3^m different temporal (s', z)-paths at time step three. Each path must visit the clause vertices $c_1, \ldots, c_m$ in the given order by construction.

Finally, we set $\Delta = 1$. This finishes the construction, for a visualization see Figure 3.3. Note that in the underlying graph the vertices c_i corresponding to the clauses have degree six and the vertices corresponding to variables as well as all auxiliary vertices have degree at most four. Further, it is easy to check that every edge in the constructed temporal graph has only one time step and that the temporal graph can be computed in polynomial time.

Correctness. We show that ϕ is satisfiable if and only if $\mathcal{G}$ has a Δ-restless temporal (s, z)-path.

($\Rightarrow$): Let us assume that there is a satisfying assignment for formula ϕ. Then we construct a Δ-restless temporal path from vertex s to z as follows. Starting from s, for each variable x_i of ϕ the Δ-restless temporal path passes through the variables $x_i^{(1)}$, $x_i^{(2)}$, $x_i^{(3)}$, and $x_i^{(4)}$, if x_i is set to false, and $\bar{x}_i^{(1)}$, $\bar{x}_i^{(2)}$, $\bar{x}_i^{(3)}$, and $\bar{x}_i^{(4)}$, if x_i is set to true, at time step one. The Δ-restless temporal path arrives at time step one in the vertex s_n. In time step two it goes from s_n to s'.

At time step three, the Δ-restless temporal path can be extended to c_1. In each

clause c_j for $j \in [m]$ there is at least one literal x_i (or $\bar{x}_i$) that is evaluated to true. Let c_j be the kth clause in which x_i appears. We have that, depending on whether x_i is set to true (or false), the vertex $x_i^{(k)}$ (or $\bar{x}_i^{(k)}$) has not been visited so far. Hence, the Δ-restless temporal path can be extended from c_j to c_{j+1} (or to z for $j = m$) at time step three via $x_i^{(k)}$ (or $\bar{x}_i^{(k)}$). Thus, there exists a Δ-restless temporal (s,z)-path in $\mathcal{G}$.

($\Leftarrow$): Let us assume that there exists a Δ-restless temporal (s,z)-path in the constructed temporal graph $\mathcal{G}$. Note that any Δ-restless temporal (s,z)-path must reach s_n in time step one because the variable gadget has only edges at time step one and the waiting time $\Delta = 1$ prevents the path to enter the clause gadget (which only has edges at time step three) before using the edge $\{s_n, s'\}$ at time step two.

It is easy to see that for the first part of the Δ-restless temporal path from s to s_n it holds that for each $i \in [n]$, it visits either vertices $x_i^{(1)}$, $x_i^{(2)}$, $x_i^{(3)}$, and $x_i^{(4)}$, or vertices $\bar{x}_i^{(1)}$, $\bar{x}_i^{(2)}$, $\bar{x}_i^{(3)}$, and $\bar{x}_i^{(4)}$. In the former case we set x_i to false and in the latter case we set x_i to true. We claim that this produces a satisfying assignment for ϕ.

In time step three, the part of the Δ-restless temporal path from s' to z has to pass vertices $c_1, c_2, \ldots, c_m$ to reach z. The Δ-restless temporal path passes exactly one variable vertex $x_i^{(k)}$ (or $\bar{x}_i^{(k)}$) when going from c_j to c_{j+1} (and finally from c_m to z) that has not been visited so far and that corresponds to a variable that appears in the clause c_j for the kth time. The fact that the variable vertex was not visited implies that we set the corresponding variable to a truth value that makes it satisfy clause c_j. This holds for all $j \in [m]$. Hence, each clause is satisfied by the constructed assignment and, consequently, ϕ is satisfiable. □

This result also implies that parameterizing Restless Temporal (s,z)-Path by the maximum degree of the underlying graph or other smaller structural graph parameters of the underlying graph cannot yield fixed-parameter tractability unless P = NP, even if combined with the lifetime ℓ.

Furthermore, the reduction presented in the proof of Theorem 3.3 yields the following running time lower bound assuming the Exponential Time Hypothesis (ETH) [IP01, IPZ01].

Corollary 3.4. Restless Temporal (s,z)-Path *does not admit an* $f(\ell)^{o(|\mathcal{G}|)}$*-time algorithm for any computable function* f *unless the ETH fails.*

Proof. First, note that any 3-SAT formula with m clauses can be transformed into an equisatisfiable Exact (3,4)-SAT formula with $O(m)$ clauses [Tov84]. The reduction presented in the proof of Theorem 3.3 produces an instance of Restless Temporal (s,z)-Path with a temporal graph of size $|\mathcal{G}| \in O(m)$ and $\ell = 3$. Hence an algorithm for Restless Temporal (s,z)-Path with running time $f(\ell)^{o(|\mathcal{G}|)}$ for some computable

function f would imply the existence of a $2^{o(m)}$-time algorithm for 3-SAT. This is a contradiction to the ETH [IP01, IPZ01]. □

We use this result in Section 3.3.3 to show that an algorithm by Casteigts et al. [Cas+20] is presumably asymptotically optimal.

3.3.2 W[1]-Hardness for Feedback Vertex Number

In the following, we show that RESTLESS TEMPORAL (s,z)-PATH is W[1]-hard when parameterized by the feedback vertex number of the underlying graph. Intuitively this means that even if the underlying graph on an input temporal graph is very similar to a forest, we presumably cannot solve the problem very efficiently (that is, in FPT-time). We show this with a parameterized reduction from MULTICOLORED CLIQUE, where we are asked whether a given k-partite graph contains a clique of cardinality k. The main idea it that we create a selection gadget for each vertex part that any restless temporal path from s to z has to traverse, where it has to visit path segments corresponding to all but one of the vertices of that part, thereby selecting one vertex. Afterwards, in a validation gadget, the restless temporal path has to traverse the non-visited path segments which is only possible if the selected vertices form a clique. As in the reduction from the previous section, the waiting times that the restless temporal path has to respect prevent it from taking "shortcuts", that is, entering the validation gadget before having traversed all vertex selection gadgets.

Theorem 3.5. RESTLESS TEMPORAL (s,z)-PATH *parameterized by the feedback vertex number of the underlying graph is W[1]-hard for all* $\Delta \geq 1$ *even if every edge has only one time stamp.*

Proof. We present a parameterized polynomial-time reduction from MULTICOLORED CLIQUE where, given a k-partite graph $H = (U_1 \uplus U_2 \uplus \ldots \uplus U_k, F)$, we are asked to decide whether H contains a clique of cardinality k. MULTICOLORED CLIQUE is known to be W[1]-hard when parameterized by the clique size k [Fel+09].

Let $(H = (U_1 \uplus U_2 \uplus \ldots \uplus U_k, F), k)$ be an instance of MULTICOLORED CLIQUE. For each $i, j \in [k]$ with $i < j$ let $F_{i,j} = \{\{u,v\} \in F \mid u \in U_i \wedge v \in U_j\}$ be the set of edges between vertices in U_i and U_j. We assume that $k \geq 3$, otherwise we can solve the instance in polynomial time. Without loss of generality, we assume that for all $i, j, i', j' \in [k]$ with $i < j$ and $i' < j'$ we have that $|F_{i,j}| = |F_{i',j'}| = m$ for some $m \in \mathbb{N}$. Note that if this is not the case, then we add new vertices and single edges to increase the cardinality of some set $F_{i,j}$ and this does not introduce new cliques since $k \geq 3$. We further assume without loss of generality that $|U_1| = |U_2| = \ldots = |U_k| = n$ for some $n \in \mathbb{N}$. If this is not the case, then we can add additional isolated vertices to increase the cardinality

of some set U_i. We construct a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ with two distinct vertices $s, z \in V$ such that there is a Δ-restless temporal (s,z)-path in $\mathcal{G}$ if and only if H contains a clique of size k. Furthermore, we show that the underlying graph $G_{\downarrow}$ of $\mathcal{G}$ has a feedback vertex number in $O(k^2)$.

Vertex Selection Gadgets. For each set U_i with $i \in [k]$ of the vertex set of H we create the following gadget. Let $U_i = \{u_1^{(i)}, u_2^{(i)}, \dots, u_n^{(i)}\}$. We create a path of length $k \cdot n + n + 1$ on fresh vertices $w_1^{(i)}, v_{1,1}^{(i)}, v_{1,2}^{(i)}, \dots, v_{1,k}^{(i)}, w_2^{(i)}, v_{2,1}^{(i)}, \dots, v_{n,k}^{(i)}, w_{n+1}^{(i)}$. Intuitively, this path contains a segment of length k for each vertex in U_i which are separated by the vertices $w_j^{(i)}$, and the construction will allow a Δ-restless temporal (s,z)-path to skip exactly one of these segments, which is going to correspond to selecting this vertex for the clique.

Formally, for each vertex $u_j^{(i)} \in U_i$ we create k vertices $v_{j,1}^{(i)}, v_{j,2}^{(i)}, \dots, v_{j,k}^{(i)}$, which we call the *segment corresponding to* $u_j^{(i)}$. We further create vertices $w_1^{(i)}, w_2^{(i)}, \dots, w_{n+1}^{(i)}$. For all $j \in [n]$ and $x \in [k-1]$ we connect vertices $v_{j,x}^{(i)}$ and $v_{j,x+1}^{(i)}$ with an edge at time $(i-1)\cdot n + j$ and we connect $w_j^{(i)}$ with $v_{j,1}^{(i)}$ and $w_{j+1}^{(i)}$ with $v_{j,k}^{(i)}$ at time $(i-1)\cdot n + j$ each.

Lastly, we introduce a "skip vertex" $s^{(i)}$ that will allow a Δ-restless temporal (s,z)-path to skip one path segment of length k that corresponds to one of the vertices in U_i. For each $j \in [n+1]$, we connect vertices $s^{(i)}$ and $w_j^{(i)}$ with an edge at time $(i-1)\cdot n + j$.

Now we connect the gadgets for all U_i's in sequence, that is, a Δ-restless temporal (s,z)-path passes through the gadgets one after another, selecting one vertex of each part U_i. Formally, for all $i \in [k-1]$, we connect vertices $w_{n+1}^{(i)}$ and $w_1^{(i+1)}$ with an edge at time $i \cdot n + 1$. It is easy to check that the set $\{s^{(1)}, s^{(2)}, \dots, s^{(k)}\}$ forms a feedback vertex set of size k for the vertex selection gadgets. The vertex selection gadget is visualized in Figure 3.4.

Validation Gadgets. A Δ-restless temporal (s,z)-path has to pass through the validation gadgets after it passed through the vertex selection gadgets. Intuitively, this should only be possible if the selected vertices form a clique. We construct the gadget in the following way.

For each $i, j \in [k]$ with $i < j$ let the edges in $F_{i,j}$ be ordered in an arbitrary way, that is, $F_{i,j} = \{e_1^{(i,j)}, e_2^{(i,j)}, \dots, e_m^{(i,j)}\}$. We create two paths of length $2m$ on fresh vertices $v_{1,1}^{(i,j)}, v_{1,2}^{(i,j)}, v_{2,1}^{(i,j)}, v_{2,2}^{(i,j)}, \dots, v_{m,2}^{(i,j)}$ and $v_{1,3}^{(i,j)}, v_{1,4}^{(i,j)}, v_{2,3}^{(i,j)}, v_{2,4}^{(i,j)}, \dots, v_{m,4}^{(i,j)}$, respectively. Intuitively, the *first path* selects an edge from U_i to U_j and the transition to the *second path* should only be possible if the two endpoints of the selected edge are selected in the corresponding vertex selection gadgets.

Formally, for each edge $e_h^{(i,j)} \in F_{i,j}$ we create four vertices $v_{h,1}^{(i,j)}, v_{h,2}^{(i,j)}, v_{h,3}^{(i,j)}, v_{h,4}^{(i,j)}$. Fur-

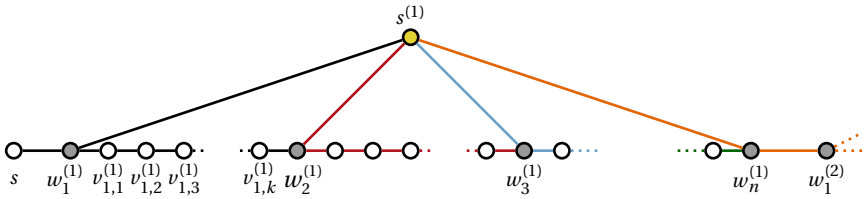


Figure 3.4: Visualization of the vertex selection gadget for U_1 from the reduction of Theorem 3.5. Black edges appear at time step one, red edges at time step two, blue edges at time step three, green edges at time step $n-1$, and orange edges at time step n. For the segment corresponding to $u_1^{(1)} \in U_1$ all vertex names are presented, for the other segments the names are analogous but omitted. The auxiliary $w_1^{(1)}, \ldots, w_n^{(1)}, \ldots$ vertices are colored gray. The "skip vertex" $s^{(1)}$ is colored yellow. Note that $\{s^{(1)}\}$ is a feedback vertex set for the vertex selection gadget for U_1.

thermore, we introduce three extra vertices $s_1^{(i,j)}, s_2^{(i,j)}, s_3^{(i,j)}$. For all $h \in [m]$ we connect vertices $v_{h,1}^{(i,j)}$ and $v_{h,2}^{(i,j)}$ with an edge at time $y_{i,j}+2h-1$, we connect vertices $v_{h,1}^{(i,j)}$ and $s_1^{(i,j)}$ with an edge at time $y_{i,j}+2h-1$, we connect vertices $v_{h,3}^{(i,j)}$ and $v_{h,4}^{(i,j)}$ with an edge at time $y_{i,j}+2h-1$, we connect vertices $v_{h,3}^{(i,j)}$ and $s_3^{(i,j)}$ with an edge at time $y_{i,j}+2h-1$, and if $h<m$, then we connect vertices $v_{h,2}^{(i,j)}$ and $v_{h+1,1}^{(i,j)}$ with an edge at time $y_{i,j}+2h$ and we connect vertices $v_{h,4}^{(i,j)}$ and $v_{h+1,3}^{(i,j)}$ with an edge at time $y_{i,j}+2h$, where $y_{i,j} = k \cdot n + 2m \cdot (i \cdot j + \frac{1}{2} \cdot i \cdot (i-1) - 1)$ (the value of $y_{i,j}$ can be interpreted as a "time offset" for the validation gadget for $F_{i,j}$, the value is computed by adding all time steps needed in validation gadget for $F_{i',j'}$ with $i' < j'$, $i' \leq i$, $j' \leq j$, and $(i',j') \neq (i,j)$). Next, for each edge $e_h^{(i,j)} = \{u_a^{(i)}, u_b^{(j)}\} \in F_{i,j}$ we connect vertices $s_1^{(i,j)}$ and $v_{a,j}^{(i)}$ (from the vertex selection gadget for U_i) with an edge at time $y_{i,j}+2h-1$, we connect vertices $s_2^{(i,j)}$ and $v_{a,j}^{(i)}$ with an edge at time $y_{i,j}+2h-1$, we connect vertices $s_2^{(i,j)}$ and $v_{b,i}^{(j)}$ (from the vertex selection gadget for U_j) with an edge at time $y_{i,j}+2h-1$, and we connect vertices $s_3^{(i,j)}$ and $v_{b,i}^{(j)}$ with an edge at time $y_{i,j}+2h-1$.

Now we connect the gadgets for all $F_{i,j}$'s in sequence, that is, a Δ-restless temporal (s,z)-path passes through the gadgets one after another, selecting one edge of each part $F_{i,j}$ of the edge set F. Formally, for each $i,j \in [k]$ with $i<j$, if $i<j-1$, then we connect vertices $v_{m,4}^{(i,j)}$ and $v_{1,1}^{(i+1,j)}$ with an edge at time $y_{i+1,j}$, and if $i=j-1<k-1$, then we connect vertices $v_{m,4}^{(i,j)}$ and $v_{1,1}^{(1,j+1)}$ with an edge at time $y_{1,j+1}$. It is easy to check that the set $\{s_1^{(1,2)}, s_2^{(1,2)}, s_3^{(1,2)}, s_1^{(1,3)}, \ldots, s_3^{(1,k)}, \ldots s_3^{(k-1,k)}\}$ forms a feedback vertex set

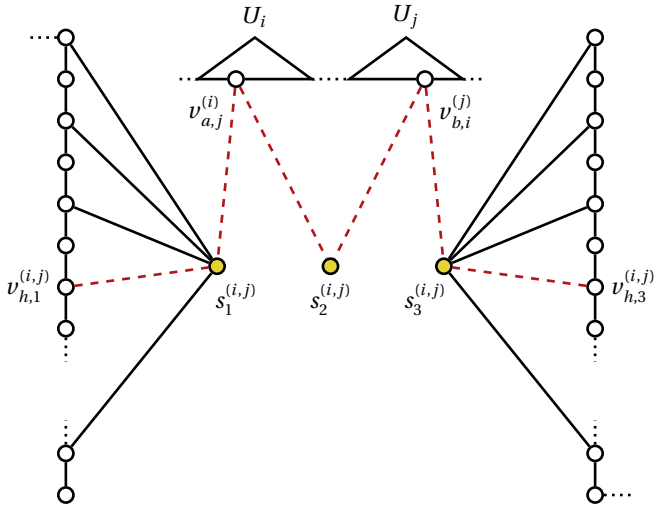


Figure 3.5: Visualization of the validation gadget for $F_{i,j}$ from the reduction of Theorem 3.5. The "first path" of the gadget is depicted vertically on the left, the "second path" on the right. The connections to the vertex selection gadgets for the edge $e_h^{(i,j)} = \{u_a^{(i)}, u_b^{(j)}\} \in F_{i,j}$ are depicted. The edges in dashed red correspond to the path through the gadget if edge $e_h^{(i,j)}$ is "selected" and all these edges have the same time stamp. The vertex selection gadgets corresponding to U_i and U_j are depicted as triangles in the upper center part. The three vertices $s_1^{(i,j)}$, $s_2^{(i,j)}$, and $s_3^{(i,j)}$ are colored yellow. Note that they form a feedback vertex set for the validation gadget for $F_{i,j}$.

of size $3 \cdot \binom{k}{2}$ for the validation gadgets. For an illustration see Figure 3.5.

Finally, we create two new vertices s and z, we connect vertices s and $w_1^{(1)}$ (the "first" vertex of the vertex selection gadgets) with an edge at time one, we connect vertices s and $s^{(1)}$ (the "skip vertex" of the first vertex selection gadget) with an edge at time one, and we connect z and $v_{m,4}^{(k-1,k)}$ (the "last" vertex of the validation gadgets) with an edge at time $k \cdot n + m \cdot (3k^2 + 5k + 3)$, which is the time that a Δ-restless temporal (s,z)-path needs to pass through all gadgets. We further connect vertices $w_{n+1}^{(k)}$ and $v_{1,1}^{(1,2)}$ (connecting the vertex selection gadgets and the validation gadgets) with an edge at time $k \cdot n$. Finally, we set $\Delta = 1$. This completes the construction. It is easy to check that $\mathcal{G}$ can be constructed in polynomial time and that the feedback vertex number of $G_!$ is at most $k + 3 \cdot \binom{k}{2}$ and that every edge has only one time stamp.

Correctness. Now we show that H contains a clique of size k if and only if there is a Δ-restless temporal path from s to z in $\mathcal{G}$.

($\Rightarrow$): Assume that H contains a clique of size k and let $X \subseteq V(H)$ with $|X| = k$ be the set of vertices that form the clique in H. Now we show how to construct a Δ-restless temporal (s,z)-path in $\mathcal{G}$. Note that since H is k-partite, we have that $|U_i \cap X| = 1$ for all $i \in [k]$. The temporal path starts at vertex s in $\mathcal{G}$ and then first passes through the vertex selection gadgets. If $u_j^{(i)} \in X$ for some $i \in [k]$ and $j \in [n]$, then the temporal path skips the segment corresponding to $u_j^{(i)}$ in the vertex selection gadget for U_i. More formally, the temporal path follows the vertices $w_1^{(i)}, v_{1,1}^{(i)}, v_{1,2}^{(i)}, \ldots, v_{1,k}^{(i)}, w_2^{(i)}, \ldots, v_{j-1,k}^{(i)}, w_j^{(i)}, s^{(i)}, w_{j+1}^{(i)}, v_{j+1,1}^{(i)}, \ldots, v_{n,k}^{(i)}, w_{n+1}^{(i)}$ in that order, that is, it skips vertices $v_{j,1}^{(i)}, v_{j,2}^{(i)}, \ldots, v_{j,k}^{(i)}$. It is easy to check that the time labels of the edges in the vertex selection gadget allow for a restless temporal path as described that respects the waiting time Δ.

In the validation gadget for $F_{i,j}$ with $i < j$, the path "selects" the edge $(U_i \cap X) \cup (U_j \cap X) \in F_{i,j}$ that connects the vertices from the parts U_i and U_j that are contained in the clique X. Let $(U_i \cap X) \cup (U_j \cap X) = \{u_a^{(i)}, u_b^{(j)}\} = e_h^{(i,j)} \in F_{i,j}$. Formally, the path follows vertices $v_{1,1}^{(i,j)}, v_{1,2}^{(i,j)}, v_{2,1}^{(i,j)}, v_{2,2}^{(i,j)}, \ldots, v_{h,1}^{(i,j)}, s_1^{(i,j)}, v_{a,j}^{(i)}, s_2^{(i,j)}, v_{b,i}^{(j)}, s_3^{(i,j)}, v_{h,4}^{(i,j)}, v_{h+1,3}^{(i,j)}, v_{h+1,4}^{(i,j)}, \ldots, v_{m,4}^{(i,j)}$ in that order. Note that vertices $v_{a,j}^{(i)}$ and $v_{b,i}^{(j)}$ have not been used by the path in the vertex selection gadgets, since they appear in the segments that were skipped by the temporal path in the corresponding vertex selection gadgets. Furthermore, since the clique in H only contains one edge that connects vertices from U_i and U_j, the vertices $v_{a,j}^{(i)}$ and $v_{b,i}^{(j)}$ have not been used by the temporal path in an earlier validation gadget. It is easy to check that the time labels of the edges in the validation gadget allow for a Δ-restless temporal path as described. After the last validation gadget the path arrives at vertex z. Hence, we have found a Δ-restless temporal (s,z)-path in $\mathcal{G}$.

($\Leftarrow$): Assume that we are given a Δ-restless temporal (s,z)-path in $\mathcal{G}$. We now show that H contains a clique of size k.

After starting at s, the Δ-restless temporal path first passes the vertex selection gadgets. Here, we need to make the important observation that for each $i \in [k]$, any Δ-restless temporal (s,z)-path has to "skip" at least one segment corresponding to one vertex $u_j^{(i)} \in U_i$ in the vertex selection gadget corresponding to U_i, otherwise the temporal path cannot traverse the validation gadgets. More formally, assume for the sake of contradiction that there is a Δ-restless temporal (s,z)-path and an $i \in [k]$ such that the temporal path visits all vertices in the vertex selection gadget corresponding to U_i. Let $j \in [k]$ with $j \neq i$. Assume that $i < j$ (the other case works analogously). We claim that the temporal path cannot traverse the validation gadget for $F_{i,j}$. For the

temporal path to go from $s_1^{(i,j)}$ to $s_2^{(i,j)}$ by construction it has to visit at least one vertex from the vertex selection gadget for U_i. If all vertices have already been visited, then this would mean that the Δ-restless temporal (s,z)-path visits one vertex twice—a contradiction.

The waiting time Δ prevents the temporal path from "skipping" more than one segment. More formally, any Δ-restless temporal (s,z)-path arrives at the "skip vertex" $s^{(i)}$ of the vertex selection gadget for U_i at time $(i-1)\cdot n+j$, for some $j \in [k-1]$. By construction this means that the path visits $w_j^{(i)}$, then $s^{(i)}$, and then has to continue with $w_{j+1}^{(i)}$ since there is only one time edge the path can use without violating the waiting time Δ. It follows that the temporal path skips exactly the segment corresponding to $u_j^{(i)} \in U_i$.

This implies that any Δ-restless temporal (s,z)-path that traverses the vertex selection gadgets leaves exactly one segment of every vertex selection gadget unvisited. Let the set $X = \{u_j^{(i)} \in U_i \mid i \in [k] \wedge j \in [n] \wedge v_{j,1}$ is an unvisited vertex$\}$ be the set of vertices corresponding to the segments that are "skipped" by the given Δ-restless temporal (s,z)-path. It is easy to check that $|X| = k$. We claim that X is a clique in H.

Assume for contradiction that it is not. Then there are two vertices $u_{i'}^{(i)}, u_{j'}^{(j)} \in X$ such that the edge $\{u_{i'}^{(i)}, u_{j'}^{(j)}\}$ is not in F. Assume that $i < j$. We show that then the Δ-restless temporal (s,z)-path is not able to pass through the validation gadget for $F_{i,j}$. By assumption we have that $\{u_{i'}^{(i)}, u_{j'}^{(j)}\} \notin F_{i,j}$. Note that the validation gadget is designed in a way that the first path "selects" an edge from $F_{i,j}$ and then the waiting time of one enforces that a Δ-restless temporal (s,z)-path can only move from the first path to the second path of a validation gadget if the two endpoints of the selected edge are vertices whose corresponding segments in the vertex selection gadget were skipped. We have seen that for every U_i with $i \in [k]$, the path segment corresponding to exactly one vertex of that set was skipped. Since $\{u_{i'}^{(i)}, u_{j'}^{(j)}\} \notin F_{i,j}$, we have that for every edge in $F_{i,j}$ the segment corresponding to at least one of the two endpoints of the edge was *not* skipped. Hence, we have that the Δ-restless temporal path cannot pass through the validation gadget of $F_{i,j}$ and cannot reach z—a contradiction. □

This result shows that parameterizing Restless Temporal (s,z)-Path with the feedback vertex number of the underlying graph and all smaller structural graph parameters of the underlying graph presumably does not yield fixed-parameter tractability. In the next section we show based on a known result that if we choose another structural graph parameter of the underlying graph, namely the treedepth (for a definition see Section 2.4) which is incomparable to the feedback vertex number, we can obtain fixed-parameter tractability.

3.3.3 Optimal Restless Temporal Paths

In this section, we discuss how to find optimal restless temporal paths. Of course, we have by Theorem 3.3 that it is NP-hard to find optimal restless temporal paths for all optimality criteria introduced in Section 3.2.4. However, the problem of finding a restless temporal path with a certain quality introduces a natural parameter to the problem, which we want to discuss here.

From Theorem 3.3 we can deduce that there is not much hope for fast or foremost restless temporal paths since the instance constructed in the reduction has lifetime $\ell = 3$ and hence the duration as well as the arrival time of any restless temporal path in this instance is at most three. However, we can observe that in all our reductions (the one of Theorem 3.3 and the one of Theorem 3.5) the length of Δ-restless temporal (s,z)-paths is unbounded. We defined Short Restless Temporal (s,z)-Path as the problem of finding restless temporal paths of bounded length, and indeed it is known that Short Restless Temporal (s,z)-Path is fixed-parameter tractable when parameterized by the length bound k [Cas+20]. The algorithm of Casteigts et al. [Cas+20] has a single-exponential running time in k, hence Corollary 3.4 implies that their algorithm is presumably asymptotically optimal. Furthermore, Proposition 3.2 shows that their fixed-parameter tractability result presumably cannot be improved to yield a polynomial kernel.

The fixed-parameter tractability result of Casteigts et al. [Cas+20] has some interesting implications for structural parameterizations. We can observe that any path of a graph can contain at most twice as many vertices as the vertex cover number of the graph (plus one) since we cannot visit two vertices outside of the vertex cover directly one after another. Essentially the same observation can be made in the temporal setting. If we consider the vertex cover number $\mathrm{vc}_\downarrow$ of the underlying graph, then we can deduce that any restless temporal path can have length at most $2\mathrm{vc}_\downarrow + 1$. From a classification standpoint, we can improve this a little further by observing that the length of any restless temporal path is upper-bounded by the length of any path of the underlying graph. The length of a path in the underlying graph can be upper-bounded by $2^{O(\mathrm{td}_\downarrow)}$ [ND12], where $\mathrm{td}_\downarrow$ is the treedepth of the underlying graph (for a definition see Section 2.4). Hence, we get the following observation.

Observation 3.6. Restless Temporal (s,z)-Path *is fixed-parameter tractable when parameterized by the treedepth $\mathrm{td}_\downarrow$ of the underlying graph.*

Theorem 3.5 implies that we presumably cannot hope to improve this result much, at least not from a classification standpoint, since most structural graph parameters that are smaller than treedepth are also smaller than feedback vertex number.

3.4 Conclusion

In this chapter we have analyzed the (parameterized) computational complexity of RESTLESS TEMPORAL (s,z)-PATH. Other than its non-restless counterpart or the "walk-version", this problem turns out to be computationally hard, even in quite restricted cases. The waiting times allow us to encode computationally hard problems into RESTLESS TEMPORAL (s,z)-PATH since we have much more control over the possibilities how a restless temporal path can traverse the temporal graph to reach z from s. We remark that since deciding whether there *exists* a restless temporal path between two vertices is already NP-hard, there is no hope for polynomial-time approximation algorithms (where the canonical objective would be to optimize the optimality, such as being short, fast, or foremost) for any approximation factor unless P = NP. On the positive side, we could identify structural parameters of the underlying graph that allow for fixed-parameter algorithms.

There are a couple of canonical future research directions. The fact that we can exclude a large number of parameters as candidates to obtain further fixed-parameter tractability results motivates considering parameter combinations or further restricting the input instances. Furthermore, the reductions we use to show computational hardness produce instances where the temporal graph changes "abruptly" over time, that is, there are time steps where many edges "appear" or "disappear". If we restrict the input to graphs where changes happen slowly, then most of our hardness result do not hold in their current state and it is not clear whether it is possible to adapt them.

It is natural to also analyze other path-related problems under this restless temporal path model, such as for example finding temporal separators or computing shortest path-based centrality measures such as closeness or betweenness. In Chapter 4 we will do the former, that is, investigating the computational complexity of finding vertex separators for restless temporal paths. However, also in other contexts such as temporal graph exploration it would make sense to consider the restless temporal path model.

CHAPTER 4

Temporal Separators

Coming from the problem of finding a special type of temporal paths in a temporal graph in Chapter 3, we continue with another path-related problem. In this chapter, we investigate the computational complexity of separating two distinct vertices s and z by vertex deletion in a temporal graph, that is, destroying all temporal paths from s to z. Since, in contrast to its static counterpart, this problem is NP-hard [KKK02], it is natural to investigate whether relevant special cases exist that are computationally tractable. To this end, we study a special class of temporal graphs, so-called *temporal unit interval graphs*, which can be seen as a basic model for temporal physical proximity networks. We provide computational hardness results as well as tractability results for temporal separation on such temporal graphs, introducing a parameter to measure the "change over time" of temporal unit interval graphs.

Additionally, we explore temporal separators under the restless temporal path model we studied in Chapter 3. We investigate the computational complexity of destroying all *restless* temporal paths between two distinct vertices s and z by vertex deletion. We show that this problem is Σ_2^{P}-complete.

This chapter is based on a series of papers investigating the computational complexity of finding temporal separators [Flu+18, Flu+20b, Zsc+18, Zsc+20].

4.1 Introduction

Having the dynamics of interactions represented in the model, it is essential to adapt definitions such as connectivity and paths to respect temporal features. This directly affects the notion of *separators* in the temporal setting. Vertex separators are a fundamental primitive in static network analysis and it is well-known that they can be computed in polynomial time [AMO93, Theorem 6.8]. In contrast to the static case, Kempe, Kleinberg, and Kumar [KKK02] showed that in temporal graphs it is NP-hard to compute minimum separators.

A natural approach to tackle computational hardness is to restrict the input instances with the goal to obtain tractability results. Motivated from applications on temporal physical proximity networks, we study temporal separators on temporal unit interval graphs. Unit interval graphs capture a very simplistic notion of physical

proximity, namely of objects in one-dimensional space. We intend this model to serve as a starting point of understanding temporal graphs that have an underlying physical proximity model (or geometric intersection model). However, we can show that finding temporal separators remains NP-hard even on temporal graphs where each layer is a unit interval graph. From a motivation standpoint it is a reasonable assumption that the entities of interest do not move arbitrarily fast in the space. And indeed, putting restrictions on the "movements" of the intervals allows us to obtain tractability results. We identify a subclass of temporal unit interval graphs where finding temporal separators is polynomial-time solvable. From there we use a "distance-to-triviality" parameterization to obtain fixed-parameter tractability results for the general case.

We also analyze temporal separators under the restless temporal path model we discussed in Chapter 3. These so-called *restless temporal separators* are vertex sets that destroy all *restless* temporal paths. Recall that restless temporal paths are well suited to model the spread of certain diseases. Hence, separators for these spreading processes are naturally motivated as well. To the best of our knowledge we give the first computational complexity analysis for the problem of finding restless temporal separators.

4.1.1 Related Work

Temporal separators, as we study them in this chapter, were first investigated by Kempe, Kleinberg, and Kumar [KKK02], who proved that computing temporal (s,z)-separators is NP-hard. In contrast, Berman [Ber96] proved earlier that computing temporal (s,z)-cuts (edge deletion instead of vertex deletion) is polynomial-time solvable. There also exist other notions of temporal cuts and separators. In the context of survivability of temporal graphs, Liang and Modiano [LM17] studied cuts where an edge deletion only lasts for δ consecutive time stamps. Moreover, they studied a temporal maximum flow defined as the maximum number of sets of journeys where each two journeys in a set do not use a temporal edge within some δ time steps. A different notion of temporal flows was introduced by Akrida et al. [Akr+19a]. They showed how to compute in polynomial time the maximum amount of flow passing from a source vertex s to a sink vertex z until a given point in time.

The vertex variant of Menger's Theorem [Men27] states that the maximum number of vertex-disjoint paths from s to z equals the size of a minimum-cardinality (s,z)-separator. In static graphs, Menger's Theorem allows for finding a minimum-cardinality (s,z)-separator via maximum flow computations. However, Berman [Ber96] proved that the vertex-variant of an analogue to Menger's Theorem for temporal graphs, asking for the maximum number of (strict) temporal paths instead,

does not hold. Kempe, Kleinberg, and Kumar [KKK02] proved that the vertex-variant of the former analogue to Menger's Theorem holds true if the underlying graph excludes a fixed minor. Mertzios, Michail, and Spirakis [MMS19] proved another analogue of Menger's Theorem: the maximum number of strict temporal (s, z)-paths which never leave the same vertex at the same time equals the minimum number of vertex departure times needed to separate s from z, where a vertex departure time (v, t) is the vertex v at time point t (what we also call vertex appearance).

For related work on temporal paths and other temporal connectivity related problems, we refer to Section 3.1.1.

4.1.2 Our Contributions and Organization of the Chapter

We analyze the computational complexity of finding temporal separators on temporal unit interval graphs. We first show that the problem remains hard on temporal unit interval graphs even if the number of layers is constant. Then we give a polynomial time algorithm for a restricted version of the problem, where we require the input temporal graphs to be order-preserving. Herein, the intervals of each layer are not allowed to change their relative ordering. We generalize this algorithm by a distance-to-triviality parameterization which upper-bounds how much the interval orderings may change over time, introducing the "shuffle number" parameter for temporal unit interval graphs.

Furthermore, we analyze the computational complexity of deciding whether a temporal graph admits a restless temporal separator, and show that this problem is Σ_2^p-complete.

The chapter is organized as follows. In Section 4.2 we formally introduce all necessary concepts related to temporal separators that we need in this chapter, we formally define our problem settings and report some basic observations about the problems. In Section 4.3 we present our findings on the computational complexity of finding temporal separators in temporal unit interval graphs. Section 4.4 discusses the computational complexity of finding restless temporal separators. We conclude in Section 4.5.

4.1.3 Further Contributions of the Papers this Chapter is Based on

Zschoche et al. [Zsc+20] focus on differences in the computational complexity of finding temporal separators that destroy *strict* temporal paths and finding temporal separators that destroy *non-strict* temporal paths.[8] They show that finding temporal separators that destroy non-strict temporal paths is NP-complete even if

[8] See Section 3.2.5 for a short discussion of strict and non-strict temporal paths.

the temporal graph has lifetime two. For the strict case, they show that the problem is NP-complete if the temporal graph has lifetime at least five and polynomial time solvable otherwise. Furthermore, they prove that both variants are W[1]-hard when parameterized by the size bound of the separator and that both variants are NP-complete even if the underlying graph is planar. Using MSO formulations, they show that finding strict temporal separators is fixed-parameter tractable with respect to the lifetime of the input temporal graph if the underlying graph is planar. Finally, they introduce the concept of a *temporal core*, which is the set of vertices of a temporal graph that have at least one incident edge, that is present at some time steps but not at all time steps. They show that finding (non-strict) temporal separators is fixed-parameter tractable with respect to the size of the temporal core while finding strict temporal separators is NP-hard even if the temporal core is empty.

Fluschnik et al. [Flu+20b] focus on non-strict temporal separators and investigate the computational complexity of finding such separators in different classes of temporal graphs. They show NP-completeness even for very restricted classes of temporal graphs, such as temporal graphs that have a line graph as underlying graph, and several restrictions on how the edge set of the temporal graph may change over time. On the positive side, they show that finding temporal separators is fixed-parameter tractable with respect to vertex cover number of the underlying graph, the combination of the treedepth and the size bound of the separator, and the combination of the treewidth of the underlying graph and the lifetime.

4.2 Preliminaries

In this section, we formally introduce the most important concepts related to temporal separators and give the formal problem definitions of Temporal (s,z)-Separation and Restless Temporal (s,z)-Separation. We further discuss some basic observations for Temporal (s,z)-Separation.

4.2.1 Temporal Separators

Intuitively, a temporal separator that separates two distinct vertices s and z is a vertex set that all temporal paths from s to z have to cross, see Figure 4.1 for an example. We introduced temporal paths in the previous chapter and refer to the definition in Section 3.2.1 (Definition 3.1). Using this, we can formally define temporal (s,z)-separators as follows.

Definition 4.1 (Temporal (s,z)-Separator)**.** Let $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ be a temporal graph with $s,z \in V$. A vertex set $S \subseteq V \setminus \{s,z\}$ is a *temporal* (s,z)*-separator* for $\mathcal{G}$ if there is no temporal (s,z)-path in $\mathcal{G} - S$.

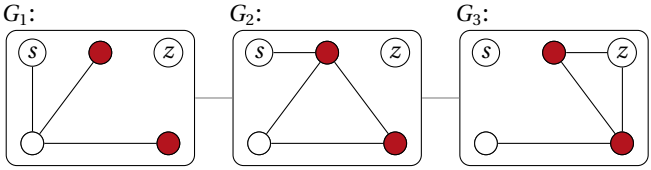


Figure 4.1: Example temporal graph with two vertices s and z and lifetime three. A temporal (s,z)-separator is depicted in red.

We can now introduce the (decision) problem of deciding whether a given temporal graph $\mathcal{G}$ with two distinct vertices s and z admits a temporal (s,z)-separator.

TEMPORAL (s,z)-SEPARATION	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, two distinct vertices $s, z \in V$, and an integer $k \in \mathbb{N}$.
Question:	Does $\mathcal{G}$ admit a temporal (s,z)-separator of size at most k?

It is easy to see that we can verify in polynomial time whether a vertex set S is a temporal (s,z)-separator for a given temporal graph $\mathcal{G}$ since we can check in polynomial time whether there is a temporal path from s to z in $\mathcal{G} - S$ [XFJ03]. Hence, we have that TEMPORAL (s,z)-SEPARATION is contained in NP.

4.2.2 Temporal Unit Interval Graphs

When we analyze the computational complexity of TEMPORAL (s,z)-SEPARATION we will focus on instances where the input temporal graph is a *temporal unit interval graph*. Recall that a static graph $G = (V,E)$ is a *unit interval graph* if there are unit-length intervals $[a_v, a_v + 1]$ with $a_v \in \mathbb{Q}$ for all vertices $v \in V$ such that for all $v, w \in V$ with $v \neq w$ we have that $\{v,w\} \in E$ if and only if $[a_v, a_v+1] \cap [a_w, a_w+1] \neq \emptyset$. A temporal unit interval graph is simply a temporal graph where every layer is a unit interval graph.

Definition 4.2 (Temporal Unit Interval Graph)**.** A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ is a *temporal unit interval graph* if for all $i \in \ell$ we have that $G_i = (V, E_i)$ is a unit interval graph.

We remark that temporal unit interval graphs can easily be recognized in polynomial time by checking whether every layer is a unit interval graph. This can be done in linear time [LO93].

4.2.3 Restless Temporal Separators

We now adapt the definition of temporal separators for restless temporal paths (which we discussed in Chapter 3), that is, a restless temporal (s,z)-separator should destroy all restless temporal (s,z)-paths in a given temporal graph. Recall that, intuitively, restless temporal paths may only "wait" a bounded number of time steps in any vertex. For the formal definition of restless paths we refer to Section 3.2.2 (Definition 3.2). Using this, we can formally define Δ-restless temporal (s,z)-separators as follows.

Definition 4.3 (Δ-Restless Temporal (s,z)-Separator)**.** Let $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ be a temporal graph with $s,z \in V$. Let $\Delta \leq \ell$. A vertex set $S \subseteq V \setminus \{s,z\}$ is a Δ-*restless temporal* (s,z)-*separator* for $\mathcal{G}$ if there is no Δ-restless temporal (s,z)-path in $\mathcal{G} - S$.

We can now formally define the (decision) problem of finding a Δ-restless temporal (s,z)-separator in a given temporal graph $\mathcal{G}$ with two distinct vertices s and z.

Restless Temporal (s,z)-Separation	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, two distinct vertices $s,z \in V$, and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.
Question:	Does $\mathcal{G}$ admit a Δ-restless temporal (s,z)-separator of size at most k?

We remark that we presumably cannot verify in polynomial time whether a vertex set S is a Δ-restless temporal (s,z)-separator for a given temporal graph $\mathcal{G}$ since, as we showed in Chapter 3, checking whether there is a Δ-restless temporal path from s to z in $\mathcal{G} - S$ is NP-hard (Theorem 3.3). Note that Restless Temporal (s,z)-Separation with $k = 0$ is the complement of Restless Temporal (s,z)-Path. This implies that Restless Temporal (s,z)-Separation is coNP-hard for $k = 0$ and hence presumably *not* contained in NP. As we will discuss later in this chapter in Section 4.4, we can indeed show that Restless Temporal (s,z)-Separation is located in the second level of the polynomial time hierarchy.

4.2.4 Basic Observations

It is known that Temporal (s,z)-Separation is already NP-complete [KKK02] and W[1]-hard when parameterized by the separator size k [Zsc+20], however we can observe that with some straightforward transformations, these hardness results hold even when every layer only contains one edge. We show that by giving a description on how to transform arbitrary instances of Temporal (s,z)-Separation to equivalent instances with the above property. The rough idea is to "stretch" the

lifetime of the temporal graph sufficiently much such that each edge has its own layer, and then repeating certain layers in a way that all temporal paths of the original instances are preserved and no new ones are introduced.

Proposition 4.1. TEMPORAL (s,z)-SEPARATION *is NP-complete and W[1]-hard when parameterized by the separator size k even if each layer has at most one edge.*

Proof. We present a simple parameterized polynomial-time reduction from TEMPORAL (s,z)-SEPARATION on general temporal graphs, which is known to be NP-complete [KKK02] and W[1]-hard when parameterized by the separator size k [Zsc+20], to TEMPORAL (s,z)-SEPARATION on temporal graphs that have at most one edge in each layer.

Construction. Let $(\mathcal{G} = (V, (E_i)_{i\in[\ell]}), s, z, k)$ be an instance of TEMPORAL (s,z)-SEPARATION. We assume without loss of generality that for all $i \in [\ell]$ we have that $E_i \neq \emptyset$. Note that trivial layers can always be removed from instances of TEMPORAL (s,z)-SEPARATION in some preprocessing step since they never create or remove temporal paths. We construct a new temporal graph $\mathcal{G}' = (V, (E'_i)_{i\in[\ell']})$ from $\mathcal{G}$ by creating for each layer $G_i = (V, E_i)$ of $\mathcal{G}$ a temporal graph $\mathcal{G}_i^{|E_i|}$ such that there is a temporal path in $\mathcal{G}_i^{|E_i|}$ if and only if there is a path in layer i of $\mathcal{G}$.

For each layer G_i of $\mathcal{G}$ we first construct a temporal graph $\mathcal{G}_i = (V, (E_j^{(i)})_{j\in[\ell_i]})$ with $\ell_i = |E_i|$ by fixing an arbitrary total order on the edge set $E_i = \{e_1, e_2, \ldots, e_{|E_i|}\}$ of G_i and setting $E_j^{(i)} = \{e_j\}$ for all $j \in [|E_i|]$. Now, we set $\mathcal{G}' = \mathcal{G}_1^{|E_1|} \circ \mathcal{G}_2^{|E_2|} \circ \ldots \circ \mathcal{G}_\ell^{|E_\ell|}$. This is obviously a polynomial-time construction. Since, for all $i \in [\ell]$ we have that $|E_i| \leq |V|^2$ and each $\mathcal{G}_i$ has $|E_i|$ many layers, we know that $\ell' \leq \ell \cdot |V|^4$. The new instance of TEMPORAL (s,z)-SEPARATION is now $(\mathcal{G}', s, z, k)$. By construction each layer of $\mathcal{G}'$ contains exactly one edge.

Correctness. Let $i \in [\ell]$ and $v, w \in V$. Observe that G_i is the underlying graph of both $\mathcal{G}_i$ and $\mathcal{G}_i^{|E_i|}$. Since every temporal path is also a path in the underlying graph, it is easy to see that for each temporal (v,w)-path in $\mathcal{G}_i^{|E_i|}$ there is a (v,w)-path in layer G_i which visits the vertices in the same order. We claim that for each (v,w)-path P of length n in layer G_i there is a temporal (v,w)-path in $\mathcal{G}_i^n$ which visits the vertices in the same order. Let $V(P) = \{v = v_0, v_1, \ldots, v_n = w\}$ such that v_j is visited before v_{j+1}, for all $0 \leq j \leq n$. We prove the claim by induction on n. If $n = 1$, then we know that there is a time edge between v and w in $\mathcal{G}_1$. For the induction step we observe that there is a time edge between $v = v_0$ and v_1 in $\mathcal{G}_1$ and, by the induction hypothesis, there is a temporal (v_1, w)-path of length $n-1$ in $\mathcal{G}_i^{n-1}$ which visits the vertices in the same order as P. Since $n \leq |E_i|$, we have that for each (v,w)-path in layer G_i there is a temporal (v,w)-path in $\mathcal{G}_i^{|E_i|}$ which visits the vertices in the

same order, where $v, w \in V$ and $i \in [\ell]$. If follows that a vertex set $S \subseteq V \setminus \{s, z\}$ is a temporal (s, z)-separator in $\mathcal{G}$ if and only if S is a temporal (s, z)-separator in $\mathcal{G}'$, because in the construction of $\mathcal{G}'$ we replaced layer G_i with $\mathcal{G}_i^{|E_i|}$. □

Proposition 4.1 also has some noteworthy implications from the point of view of parameterized complexity: Parameterizing Restless Temporal (s, z)-Separation by structural graph parameters of the layers of the input temporal graph that are constant on graphs with only one edge cannot yield fixed-parameter tractability unless P = NP, even if combined with k.

For the problem Restless Temporal (s, z)-Separation, it is easy to observe that computational hardness for some fixed value of Δ implies hardness for all larger values of Δ. This allows us to construct hardness reductions for small fixed values of Δ and still obtain general hardness results.

Observation 4.2. *For every fixed* Δ, Restless Temporal (s, z)-Separation *on instances* $(\mathcal{G}, s, z, k, \Delta + 1)$ *is computationally at least as hard as* Restless Temporal (s, z)-Separation *on instances* $(\mathcal{G}, s, z, k, \Delta)$.

This follows from an argument analogous to the proof of Observation 3.1, which says that we have essentially the same property for Restless Temporal (s, z)-Path.

Finally, we can also observe the following kernelization lower bound for Restless Temporal (s, z)-Separation.

Observation 4.3. Restless Temporal (s, z)-Separation *parameterized by the number* $|V|$ *of vertices does not admit a polynomial kernel for all* $\Delta \geq 1$ *unless NP* $\subseteq$ *coNP/poly.*

This follows directly from the result of Chapter 3 that Restless Temporal (s, z)-Path does not admit a polynomial kernel when parameterized by the number $|V|$ of vertices for all $\Delta \geq 1$ unless NP $\subseteq$ coNP/poly (Proposition 3.2). This follows from the observation that Restless Temporal (s, z)-Separation with $k = 0$ is the complement of Restless Temporal (s, z)-Path and hence a polynomial kernel for Restless Temporal (s, z)-Separation parameterized by $|V|$ would also be a polynomial kernel for Restless Temporal (s, z)-Path parameterized by $|V|$.

4.3 Separators in Temporal Unit Interval Graphs

In this section we investigate the computational complexity of Temporal (s, z)-Separation for the case where the input temporal graph is restricted to be a temporal unit interval graph. Note that, by Proposition 4.1, Temporal (s, z)-Separation

is NP-complete under this restriction. However, the construction from the proof of Proposition 4.1 increases the number of layers quite heavily. In the following, we first show that TEMPORAL (s,z)-SEPARATION is NP-complete on temporal unit interval graphs even if the number of layers is constant. Afterwards we show how to put additional restrictions on the input instances to obtain tractability results.

4.3.1 NP-Hardness for Few Layers

Zschoche et al. [Zsc+20, Theorem 3.1] showed by a reduction from VERTEX COVER that TEMPORAL (s,z)-SEPARATION is NP-complete for all $\ell \geq 2$, even if the underlying graph has constant degeneracy and every edge appears in at most one layer. In this section, we show how to modify this reduction to show that TEMPORAL (s,z)-SEPARATION is NP-complete on temporal unit interval graphs even if the input temporal graph has only six layers, thus strengthening the NP-hardness result for TEMPORAL (s,z)-SEPARATION on temporal unit interval graphs that is implied by Proposition 4.1. We remark that this leaves the obvious question of determining the computational complexity of the problem on temporal unit interval graphs with $1 < \ell < 6$.

The main idea behind the reduction from VERTEX COVER that we use to show the discussed computational hardness result is to create a gadget for each vertex such that one can use two types of vertex sets to separate s from z in this gadget: a small one and a large one. Then, for each edge in the VERTEX COVER instance, we connect the corresponding gadgets in such a way that at least in one of the gadgets it is necessary to take the large vertex set. Hence, taking the large vertex set from a gadget into the temporal (s,z)-separator corresponds to taking the vertex into the vertex cover.

Theorem 4.4. TEMPORAL (s,z)-SEPARATION *on temporal unit interval graphs is NP-complete for all* $\ell \geq 6$.

Proof. We present a polynomial-time reduction from the NP-complete VERTEX COVER problem [GJ79, Kar72] to TEMPORAL (s,z)-SEPARATION on temporal unit interval graphs with lifetime six (for larger lifetimes we can append trivial layers). Our reduction is a modification of a reduction developed by Zschoche et al. [Zsc+20, Theorem 3.1]. Recall that in VERTEX COVER, we are given a graph H and an integer h and we are asked to select at most h vertices from H such that every edge in H contains at least one selected vertex.

Construction. Let $(H = (U,F), h)$ be an instance of VERTEX COVER. We construct a TEMPORAL (s,z)-SEPARATION instance $(\mathcal{G} = (V, (E_i)_{i\in[\ell]}), s, z, |U| + h)$ with $\ell = 6$ with

$$V := \{x, v, x_v, x'_v, x_{vw} \mid v, w \in U \wedge x \in \{s,z\}\},$$

where we assume that U does not contain vertices called s or z. To define the edge sets of $\mathcal{G}$ we first define, for any $x, y \in \{s, z\}$, the following four edge classes

$$E_\alpha(x) := \binom{\{x, x_v, x'_v \mid v \in U\}}{2},$$
$$E_\beta(x) := \bigcup_{v \in U} \binom{\{x_v, x'_v, x_{vv}\}}{2} \cup \binom{\{x_{vw} \mid w \in U\}}{2},$$
$$E_\gamma(x, y) := \bigcup_{v \in U} \{\{x_v, x'_v\}, \{v, x_v\}, \{v, x'_v\}, \{v, y_{vv}\}\}, \text{ and}$$
$$E_\delta := \{\{s_{vw}, z_{wv}\}, \{s_{wv}, z_{vw}\} \mid \{v, w\} \in F\}.$$

Now we define the edge sets of $\mathcal{G}$ in the following way.

$$E_1 := E_\alpha(s),$$
$$E_2 := E_\beta(s),$$
$$E_3 := E_\gamma(z, s) \cup E_\delta,$$
$$E_4 := E_\gamma(s, z),$$
$$E_5 := E_\beta(z), \text{ and}$$
$$E_6 := E_\alpha(z).$$

The underlying graph of the constructed temporal graph $\mathcal{G}$ is visualized in Figure 4.2. It is easy to see that the construction can be done in polynomial time.

Temporal unit interval property. To see that each layer of $\mathcal{G}$ is in fact a unit interval graph, first observe that $E_\gamma(z, s)$ and E_δ are vertex-disjoint and thus each connected component of each layer is taken from a single edge class. Furthermore, for any choice $x, y \in \{s, z\}$ we have that

- $E_\alpha(x)$ forms a clique of size $2|U| + 1$,
- each connected component of $E_\beta(x)$ consists of a triangle and a clique of size $|U|$ that share exactly one vertex,
- each connected component of $E_\gamma(x, y)$ is the union of a triangle and a single edge, joined on a common vertex, and
- E_δ is a disjoint union of edges.

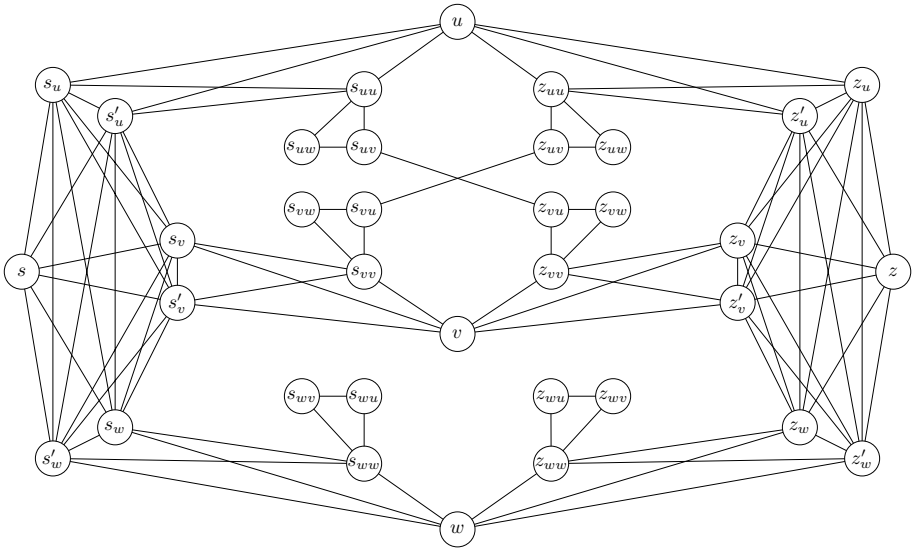


Figure 4.2: Underlying graph of the constructed temporal graph $\mathcal{G}$ resulting from a VERTEX COVER instance $H = (U, F)$ on three vertices $U = \{u, v, w\}$ and one edge $F = \{\{u, v\}\}$. The two time edges of $\mathcal{G}$ corresponding to $\{u, v\} \in F$ appear as edges $\{s_{uv}, z_{vu}\}$ and $\{s_{vu}, z_{uv}\}$ in the underlying graph (the two crossing edges in the top half of the figure).

In summary, each connected component of each layer is either a clique or a union of two cliques sharing a single vertex and thus it is an interval graph.

Correctness. Observe that no temporal (s, z)-path can use more than one edge from E_δ as it would need to use an edge from E_β in between. Consequently we may assume that any minimum temporal (s, z)-separator only contains vertices from the set $\{v, s_{vv}, z_{vv} \mid v \in U\}$ as we could exchange any other vertex for one of these. After these observations the rest of the correctness proof works analogously to the proof of Zschoche et al. [Zsc+20, Proposition 3.2]. For the sake of self-containedness we give the proof here.

($\Rightarrow$): Let $X \subseteq U$ be a vertex cover of size at most h for H. We claim that $S := (U \setminus X) \cup \{s_{vv}, z_{vv} \mid v \in X\}$ is a temporal (s, z)-separator for $\mathcal{G}$. There are $|U| - |X|$ vertices not in the vertex cover X and for each of them there is exactly one vertex in S. For each vertex in X there are two vertices in S. Hence, we have that $|S| \leq |U| - h + 2h = |U| + h$.

Assume for contradiction that there is a temporal (s, z)-path P in $\mathcal{G} - S$. Consider

the first time edge used by P that has a time stamp larger than one. Without loss of generality we can differentiate between two scenarios:

1. Either P arrives at vertex v for some $v \in U$ after traversing the first time edge with a time stamp larger than one,

2. or P arrives at vertex s_{vv} for some $v \in U$ after traversing the first time edge with a time stamp larger than one.

The first case can easily be excluded: Note that by construction of $\mathcal{G}$ we have that P arrives at v via a time edge with time stamp four and the only way to continue the path is using the edge $\{v, z_{vv}\}$ which also has time stamp four. It is easy to check that P does not go "back" to s_v or s'_v. However, we have that either v or z_{vv} is contained in S for all $v \in U$. Hence, we can conclude that P cannot arrive at z via v.

In the second case we have by construction of $\mathcal{G}$ that P arrives at S_{vv} via a time edge with time stamp two. Again, it is easy to check that P does not go "back" to s_v or s'_v. By essentially the same argument as for the first case, we can conclude that P does not continue to v. The only other vertices adjacent to s_{vv} are vertices s_{vw} for $w \in V$ and $v \neq w$. Without loss of generality assume that P out of all vertices adjacent to s_{vv} visits s_{vw} last. The only way to continue the path with a vertex that is not adjacent to s_{vv} is to visit z_{wv} next. However, notice that by construction s_{vw} and z_{wv} are only adjacent if $\{v, w\} \in F$ and, if this is the case, then we have that either s_{vv} or z_{ww} is contained in S. Clearly, $s_{vv} \in S$ is a contradiction to P being a temporal (s, z)-path in $\mathcal{G} - S$, so let us assume that $z_{ww} \in S$. However, then the only vertices that P could still visit are vertices z_{wu} for $u \in U$ with $u \neq w$. Hence, P can never reach z—a contradiction.

($\Leftarrow$): Let S be a temporal (s, z)-separator in $\mathcal{G}$ of size at most $k = |U| + h$ and let $v \in U$. Recall that there are two disjoint temporal (s, z)-separators in the vertex gadget of v, namely $\{v\}$ and $\{s_{vv}, z_{vv}\}$, and that all vertices in $V \setminus \{s, z\}$ are from a vertex gadget. We start with a preprocessing to ensure that for each vertex gadget only one of these two separators are in S. Let $S_v = S \cap \{v, s_{vv}, z_{vv}\}$. We iterate over S_v for each $v \in U$:

Case 1: If $S_v = \{v\}$ or $S_v = \{s_{vv}, z_{vv}\}$, then we do nothing.

Case 2: If $S_v = \{v, s_{vv}, z_{vv}\}$, then we remove v from S. One can observe that all temporal (s, z)-paths which are visiting v are still separated by s_{vv} or z_{vv}.

Case 3: If $S_v = \{v, s_{vv}\}$, then we remove v from S and add z_{vv}. One can observe that S is still a temporal (s, z)-separator of size at most k in $\mathcal{G}$.

Case 4: If $S_v = \{v, z_{vv}\}$, then we remove v from S and add s_{vv}. One can observe that S is still a temporal (s,z)-separator of size at most k in $\mathcal{G}$.

Case 5: If $S_v = \emptyset$, then we have that $s_v, s'_v \in S$ or $z_v, z'_v \in S$. We remove s_v, s'_v, z_v, z'_v from S and add s_{vv} and z_{vv}. One can observe that S is still a temporal (s,z)-separator of size at most k in $\mathcal{G}$.

This is a complete case distinction because neither $\{s_{vv}\}$ nor $\{z_{vv}\}$ separate all temporal (s,z)-paths in the vertex gadget of v. Now we construct a vertex cover X for H by taking $v \in U$ into X if $\{s_{vv}, z_{vv}\} \subseteq S$. Since there are $|U|$ vertex gadgets in $\mathcal{G}$ each containing either one or two vertices from S, it follows that $|X| = |S| - |U| \leq h$.

Assume towards a contradiction that X is not a vertex cover of H. Then there is an edge $\{v,w\} \in F$ where $v, w \notin X$. Hence, $s_{vv}, z_{vv}, s_{ww}, z_{ww} \notin S$ and $v, w \in S$. This contradicts the fact that S is a temporal (s,z)-separator in $\mathcal{G}$, because

$$P = \big((\{s, s_v\}, 1), (\{s_v, s_{vv}\}, 2), (\{s_{vv}, s_{vw}\}, 2), (\{s_{vw}, z_{wv}\}, 3), \\ (\{z_{wv}, z_{ww}\}, 5), (\{z_{ww}, z_w\}, 5), (\{z_w, z\}, 6)\big)$$

is a temporal (s,z)-path in $\mathcal{G} - S$. It follows that X is a vertex cover of H of size at most h. □

Theorem 4.4 clearly shows that restricting the input instances of Temporal (s,z)-Separation to temporal unit interval graph presumably does not suffice to obtain tractability results. In particular we cannot hope for a fixed-parameter algorithm for the parameter ℓ unless P = NP. Hence, it seems that we have to put further restrictions on temporal unit interval graphs. In the next section, we discuss one promising way to do this.

4.3.2 (Almost) Order-Preserving Temporal Unit Interval Graphs

In this section, we analyze how we further restrict temporal unit interval graphs such that we can obtain tractability results for Temporal (s,z)-Separation on these graphs. In particular, we restrict how much the intervals may change over time. This is a layer-wise restriction with, additionally, a temporal restriction. Recall from Theorem 4.4 that Temporal (s,z)-Separation remains NP-complete on temporal graphs where each layer is a unit interval graph, even if the lifetime ℓ is a small constant.

Now we show that if there is an ordering on the vertices that matches the relative positions of the intervals in all layers, then we can solve Temporal (s,z)-Separation in polynomial time. We then generalize this by introducing a parameter that,

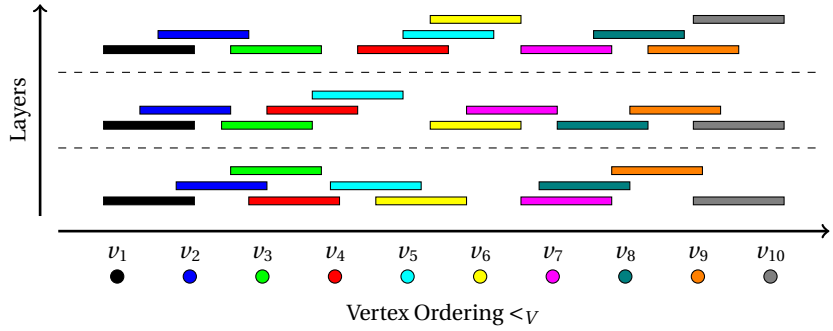


Figure 4.3: Visualization of an order-preserving temporal unit interval graph with three layers (separated by dashed lines) and vertices $\{v_1, v_2, \ldots, v_{10}\}$. The intervals corresponding to the vertices are visualized horizontally in the same color as the vertices they correspond to. In this picture we can clearly see that the relative ordering of the intervals in the three layers is the same and is compatible with the vertex ordering $<_V$ illustrated on the bottom.

informally speaking, describes how much the interval orderings may change over time, and show fixed-parameter tractability with respect to the combination of this new parameter and the lifetime ℓ.

We call a total ordering $<_V$ on a vertex set V *compatible* with a unit interval graph $G = (V, E)$ if there are unit intervals $[a_v, a_v + 1]$ with $a_v \in \mathbb{Q}$ for all vertices $v \in V$ that induce the graph G and for all $u, v \in V$ with $u <_V v$ we have that $a_u \leq a_v$. Note that for every unit interval graph there is a total ordering on the vertices that is compatible with it.

Now we use the compatibility property of a vertex ordering to define a subclass of temporal unit interval graphs that admits a vertex ordering that is compatible with every layer.

Definition 4.4. A temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ is an *order-preserving temporal unit interval graph* if $\mathcal{G}$ is a temporal unit interval graph and there is a total ordering $<_V$ on the vertex set V that is compatible with every layer G_i.

For a visualization of an order-preserving temporal unit interval graph see Figure 4.3. Given an order-preserving temporal unit interval graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$, we denote by $<_V$ a compatible total ordering on V. Let $n = |V|$, and number the vertices in $V = \{v_1, v_2, \ldots, v_n\}$ such that $v_i <_V v_j \Leftrightarrow i \leq j$. Furthermore, we use the following notation: $V_{<i} := \{v_j \mid 1 \leq j < i\}$ and $V_{>i} := \{v_j \mid n \geq j > i\}$ and $N^{>}_{G_t}(v_i) := N_{G_t}(v_i) \cap V_{>i}$. If the ordering $<_V$ is clear from the context, then we refer to vertices as smaller or larger

than other vertices to express that they appear before or after, respectively, in the ordering $<_V$.

Lemma 4.5. *Order-preserving temporal unit interval graphs can be recognized in $O(|V|^2 \cdot \ell)$ time and a compatible vertex ordering for a given order-preserving temporal unit interval graph can be computed in $O(|V|^2 \cdot \ell)$ time.*

Proof. Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a temporal graph. Then, due to Looges and Olariu [LO93, Theorem 1], we know that $\mathcal{G}$ is an order-preserving temporal unit interval graph with vertex ordering $<_V$ if and only if the vertices in every closed neighborhood $N_{G_i}[v] := N_{G_i}(v) \cup \{v\}$ with $v \in V$ of every layer $i \in [\ell]$ appear consecutively in the ordering $<_V$. Thus, the problem can be solved by searching for a column ordering of the matrix $M \in \{0,1\}^{|V| \cdot \ell \times |V|}$, defined by $M[(i,t), j] = 1 \Leftrightarrow v_j \in N_{G_t}[v_i]$, that has the *consecutive ones property*, that is, for each column we have that all ones in this column appear consecutively. It is known that this task can be solved in linear time [BL76]. □

We now state some useful properties of temporal paths and separators in order-preserving temporal unit interval graphs. Due to (3) of the following lemma, we will henceforth assume without loss of generality that $v_1 = s$ and $v_n = z$.

Lemma 4.6. *Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be an order-preserving temporal unit interval graph with ordering $<_V$.*

1. *For all $1 \leq a \leq b \leq \ell$ and for all $S \subseteq V$ we have that $\mathcal{G}|_{[a,b]} - S$ is also an order-preserving temporal unit interval graph.*

2. *If for some $1 \leq i < j \leq n$ there is a temporal (v_i, v_j)-path P in $\mathcal{G}$, then there is temporal (v_i, v_j)-path P' in $\mathcal{G}$ that visits its vertices in the order given by $<_V$.*

3. *Let $S \subseteq V$ be a temporal (v_i, v_j)-separator in $\mathcal{G}$ for some $1 \leq i < j \leq n$. Then $S' := S \setminus (V_{<i} \cup V_{>j})$ is also a temporal (v_i, v_j)-separator in $\mathcal{G}$.*

4. *A temporal (v_i, v_j)-separator in $\mathcal{G}$ is also a temporal $(v_{i'}, v_{j'})$-separator in $\mathcal{G}$ for all $1 \leq i' \leq i < j \leq j' \leq n$.*

5. *Let $S \subseteq V \setminus \{s, z\}$ such that v_i is the largest vertex reachable from s in $\mathcal{G} - S$. Let t denote the first time step when v_i is reachable from s in $\mathcal{G} - S$, and let $t \leq t' \leq \ell$. Then $N^{>}_{G_{t'}}(v_i) \subseteq S$.*

6. *Let $S_1 \subseteq V \setminus \{s,z\}$ such that v_i is the largest vertex reachable from s in $\mathcal{G}|_{[t]} - S_1$ for some $t \in [\ell-1]$. Let $S_2 \subseteq V \setminus \{s,z\}$ such that v_j is the largest vertex reachable from s in $\mathcal{G}|_{[t+1,\ell]} - S_2$. If $i \leq j$, then $S := S_1 \cup S_2$ is a temporal (s,z)-separator in $\mathcal{G}$ such that there is no vertex reachable from s in $\mathcal{G} - S$ that is larger than v_j.*

7. *Let $S \subseteq V$ be an inclusion-wise minimal temporal (s,z)-separator in $\mathcal{G}$ with the property that a given v_i is the largest vertex that is reachable from s in $\mathcal{G} - S$ and let v_j be the smallest vertex that is not in S such that S is also a temporal (s,v_j)-separator in $\mathcal{G}$. Then, for all $v_i <_V v <_V v_j$ with $v_i \neq v \neq v_j$, we have that $v \in S$, and we have that $S \cap V_{>j} = \emptyset$.*

Proof. (1): Obvious.

(2): We prove that there is a temporal (v_i, v_j)-path P' in $\mathcal{G}$ that visits its vertices in the order given by $<_V$ and $t \leq t'$, where t and t' denote the first time label in P and in P', respectively. We give an inductive proof over the number of edges in the temporal (v_i, v_j)-path P. For the base case, if P has only one edge, then $E(P) = (\{v_i, v_j\}, t)$ for some $t \in [\ell]$. Hence, $P' := P$ clearly is the sought temporal path. Now, assume that the statement holds for all temporal (v_i, v_j)-paths with at most $\ell \in \mathbb{N}$ edges. For the inductive step, let P be a temporal (v_i, v_j)-path with exactly $\ell+1$ edges. Let $v_{i'}$ be the last vertex on P such that $i' \leq i$, and let $t \in [\ell]$ be the index of the layer where P contains the edge $\{v_{i'}, v_x\}$, where v_x is the successor of $v_{i'}$ on P. Since G_t is a unit interval graph with order $<_V$, the edge $\{v_i, v_x\}$ is present in G_t. Denote by P_x the temporal (v_x, v_j)-subpath of P, starting at vertex v_x. Observe that P_x has at most ℓ edges, and hence there is a path P'_x visiting its vertices in the order given by $<_V$ and starting at some time label $t' \geq t$. Thus, the path $P' = (\{v_i\} \cup V(P'_x), \{(\{v_i, v_x\}, t)\} \cup E(P'_x))$ that starts with edge $(\{v_i, v_x\}, t)$ and then follows P'_x, visits its vertices in the order given by $<_V$ and starts at time label t being at least the first time label appearing on the edges of P.

(3): Follows directly from (2).

(4): Follows directly from (2).

(5): Suppose not. Then there is a time step t'' with a neighborhood that is not contained in S and hence there is a vertex $v_j \in N^{>}_{G_{t''}}(v_i) \setminus S$. Hence, v_j with $j > i$ is reachable from s in $\mathcal{G}_{[1:t'']} - S$, contradicting the definition of v_i.

(6): Follows directly from (2).

(7): Assume towards a contradiction that there is a vertex $v \notin S$ with $v_i <_V v <_V v_j$. Then either v is reachable from s in $\mathcal{G} - S$, which would be a contradiction to v_i being the largest vertex reachable from s in $\mathcal{G} - S$, or v is not reachable from s in $\mathcal{G} - S$, a

contradiction to the assumption that v_j is the smallest vertex such that S is also a temporal (s, v_j)-separator in $\mathcal{G}$. Furthermore, $S \cap V_{>j} = \emptyset$ follows from the assumption that S is inclusion-wise minimal and Lemma 4.6(3). □

Now we have the necessary tools to prove that Temporal (s,z)-Separation can be solved in polynomial time on order-preserving temporal unit interval graphs. The algorithm we develop to show this result uses a dynamic programming table that, intuitively, has two dimensions: vertices and time steps. Imagine a cell that corresponds to a vertex v and a time step t. In this cell, we want (informally) to save a temporal separator that separates s from all vertices that are larger than v according to the order-preserving vertex ordering if edges that appear later than t cannot be used. We fill this cell entry by finding the best possible way to extend a separator for an earlier time and an earlier vertex (according to the vertex ordering) with a separator that contains v. The former can be looked up in the table and the latter can be found by looking at the neighborhood of v in all layers up to time step t.

Formally, we show the following result.

Theorem 4.7. Temporal (s,z)-Separation *on order-preserving temporal unit interval graphs is solvable in* $O(|V|^2 \cdot \ell^2)$ *time.*

Proof. Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a given order-preserving temporal unit interval graph and k be a given upper bound on the temporal separator size. By Lemma 4.5 we can find a total vertex ordering $<_V$ compatible with every layer. We assume that there is no layer with an edge between s and z. In order to solve the problem, we use the following dynamic programming table T of size $\ell \times (n-1)$. In the table entry $T(t,i)$ we store a minimum temporal (s,z)-separator S for $\mathcal{G}|_{[t]}$ with the property that there is no vertex reachable from s in $\mathcal{G}|_{[t]} - S$ that is larger than v_i. Let

$$\mathcal{N}(v,t,t') := \begin{cases} \{N^{>}_{G_{t''}}(v) \mid t \leq t'' \leq t'\}, & \text{if } \forall\, t \leq t'' \leq t' : \{v,z\} \notin E_{t''}, \\ \{V \setminus \{s,z\}\}, & \text{otherwise.} \end{cases}$$

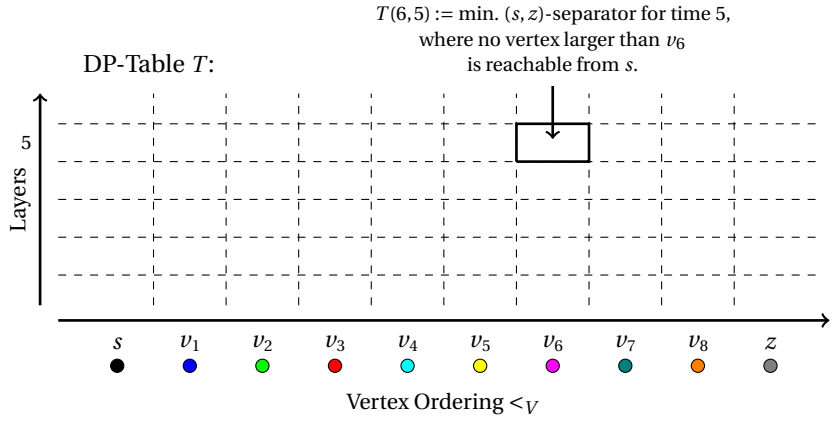


Figure 4.4: Visualization of the dynamic programming table T used in the algorithm described in the proof of Theorem 4.7.

Let T be defined in the following way:

$$T(1,1) := N_{G_1}(s), \tag{4.1}$$

$$T(t,1) := \operatorname*{arg\,max}_{S \in \mathcal{N}(s,1,t)} |S|, \tag{4.2}$$

$$T(1,i) := \operatorname*{arg\,min}_{S \in Y_i} |S|, \text{ where } Y_i := \{T(1,i-1)\} \cup \mathcal{N}(v_i,1,1), \tag{4.3}$$

$$T(t,i) := \operatorname*{arg\,min}_{S \in X_{t,i}} |S|, \text{ where} \tag{4.4}$$

$$X_{t,i} := \{ \operatorname*{arg\,max}_{S \in \mathcal{N}(v_i,t'+1,t)} |S| \cup T(t',i') \mid i' \in [i-1] \wedge t' \in [t-1] \}$$

$$\cup \{T(t,i-1)\} \cup \{ \operatorname*{arg\,max}_{S \in \mathcal{N}(v_i,1,t)} |S| \}.$$

We decide whether we face a YES-instance by checking if there is an $i \in [n-1]$ such that $|T(\ell,i)| \leq k$. For a visualization of the dynamic programming table T see Figure 4.4.

It is easy to see that each table entry can be computed in $O(|V| \cdot \ell)$ time and the table has size $|V| \cdot \ell$. Hence, the algorithm has the claimed polynomial running time.

Correctness. We prove by induction on both dimensions of T that $T(t,i)$ is a minimum temporal (s,z)-separator S for $\mathcal{G}|_{[t]}$ with the property that there is no vertex

reachable from s in $\mathcal{G}|_{[t]} - S$ that is larger than v_i with respect to $<_V$. First, observe that Lemma 4.6(5) implies that $T(1,1)$ and $T(t,1)$ are correctly filled in Equalities (4.1) and (4.2). Hence, the base for our induction is correct.

We proceed with the proof of the cases specified by Equalities (4.3) and (4.4) in two steps. First we show that for all $T(t,i)$ with $t \geq 1$ and $i > 1$, we have that $T(t,i)$ is a temporal (s,z)-separator S for $\mathcal{G}|_{[t]}$ with the property that there is no vertex reachable from s in $\mathcal{G}|_{[t]} - S$ that is larger than v_i. Then, in a second step, we show that said separator is *minimum*.

It is easy to check that if $t = 1$, then for all $i \in [n-1]$ we have that $T(1,i)$ (as specified in Equality (4.3)) is a temporal (s,z)-separator with the desired properties. Next, we consider the case that $t, i > 1$. We show that every set in $X_{t,i}$ is a temporal (s,z)-separator with the desired properties. By induction we know that this holds for $T(t,i-1)$. It is also easy to check that it holds for $S' := \arg\max_{S \in \mathcal{N}(v_i,1,t)} |S|$. For arbitrary $i' \in [i-1]$ and $t' \in [t-1]$ (Equality (4.4)) it is also straightforward to see that $S' := T(t',i') \cup \arg\max_{S \in \mathcal{N}(v_i,t'+1,t)} |S|$ has the desired properties. By induction, $T(t',i')$ contains a temporal (s,z)-separator for $\mathcal{G}|_{[t']}$ with the property that there is no vertex reachable from s in $\mathcal{G}|_{[t']} - T(t',i')$ that is larger than $v_{i'}$. The set $S'' := \arg\max_{S \in \mathcal{N}(v_i,t'+1,t)} |S|$ either equals $V \setminus \{s,z\}$, in which case we clearly have a separator with the desired properties, or it forms a temporal (s,z)-separator for $\mathcal{G}|_{[t'+1,t]}$ with the property that there is no vertex reachable from s in $\mathcal{G}|_{[t'+1,t]} - S'$ that is larger than v_i. Then by Lemma 4.6(6) we get that we have a separator with the desired properties.

Now we show that for all $t \geq 1$ and $i > 1$, the separator contained in $T(t,i)$ is of minimum size. Let $S^\star \subseteq V \setminus \{s,z\}$ be a minimum set of vertices such that in $\mathcal{G}|_{[t]} - S^\star$ the vertex v_j, $j \leq i$, is the largest reachable vertex from s. If $j < i$, then by our induction hypothesis (both for $t = 1$ and $t > 1$) we have that $|S^\star| \geq |T(t,i-1)|$ and hence $|T(t,i)| \leq |S^\star|$.

We continue with the case that $j = i$. If v_i is reachable in $\mathcal{G}|_{[1]} - S^\star$ from s, then by Lemma 4.6(5) we know that $N^{>}_{G_{t'}}(v_i) \subseteq S^\star$ for all $t' \in [t]$. As $S^\star$ is minimum, it holds that $|S^\star| = \max_{S \in \mathcal{N}(v_i,1,t)} |S|$, and we have that $\arg\max_{S \in \mathcal{N}(v_i,1,t)} |S| \in X_{t,i}$ (if $t = 1$, then $\arg\max_{S \in \mathcal{N}(v_i,1,t)} |S| \in Y_i$) which implies that $|T(t,i)| \leq |S^\star|$.

Now assume that $t > 1$ and v_i is not reachable from s in $\mathcal{G}|_{[1]} - S^\star$. Let t' be the largest time-step in which v_i is not reachable from s in $\mathcal{G}|_{[t']} - S^\star$, and let $i' < i$ be the largest index such that $v_{i'}$ is reachable from s in $\mathcal{G}|_{[t']} - S^\star$. By Lemma 4.6(5), we know that $S'' := N^{>}_{G_{t''}}(v_i)$, where $t'+1 \leq t'' \leq t$ achieves the maximum cardinality, is contained in $S^\star$. Let S' be the smallest subset of $S^\star$ such that in $\mathcal{G}|_{[t']} - S'$ the vertex $v_{i'}$ is the largest reachable vertex from s. By induction hypothesis, we have that $|S'| \geq |T(t',i')|$. From Lemma 4.6(7) it follows that $S' \cap S'' = \emptyset$. Thus, because $S^\star$ is

minimum, we can write $S^\star = S' \uplus S''$. Hence, we have

$$|S| = |S'| + |S''| \geq |T(t', i')| + |N^{>}_{G_{t''}}(v_i)| \geq \min_{S \in X_{t,i}} |S| = |T(t, i)|,$$

where the second inequality follows from the fact that $T(t', i') \cup N^{>}_{G_{t''}}(v_i) \in X_{t,i}$. □

Next we show how to use the derived polynomial-time algorithm as a basis for a distance-to-triviality parameterization [Cai03, GHN04]. For a temporal unit interval graph we introduce a parameter κ that bounds how much the compatible vertex orderings of two consecutive layers of a temporal unit interval graph differ. We use the *Kendall tau* distance [Ken38] to measure the similarity of vertex orderings. The Kendall tau distance K is a metric that counts the number of pairwise disagreements between two total orderings; it is also known as "bubble sort distance". We call the parameter κ the *shuffle number* of a temporal unit interval graph and define it as follows.

Definition 4.5 (Shuffle Number)**.** Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a temporal unit interval graph. The *shuffle number* κ of $\mathcal{G}$ is the smallest integer such that there are vertex orderings $<_V^1, <_V^2, \ldots, <_V^\ell$ with the property that $<_V^t$ is compatible with layer G_t for all $t \in [\ell]$, and the orderings of any two consecutive layers have Kendall tau distance at most κ, that is, for all $t \in [\ell - 1]$ we have that $K(<_V^t, <_V^{t+1}) \leq \kappa$. We say that the vertex orderings $<_V^1, <_V^2, \ldots, <_V^\ell$ *witness* the shuffle number of $\mathcal{G}$.

Clearly for order-preserving temporal unit interval graphs we have that $\kappa = 0$ and it is easy to observe (with a slightly modified version of Proposition 4.1, where in the construction, we add sufficiently many additional trivial layers between any pair of adjacent non-trivial layers) that we get NP-completeness for $\kappa = 1$. However, if we consider the parameter combination $(\kappa + \ell)$, then the problem becomes fixed-parameter tractable, if the vertex orderings that witness the shuffle number κ are part of the input. We need the latter, because we do not know of an efficient algorithm to compute vertex orderings that witness the shuffle number κ of a given temporal unit interval graph.

Intuitively, the algorithm we give to show this result partitions the vertices into parts that are order-preserving and into parts that are not order-preserving. In the former parts, the algorithm uses the algorithm described in the proof of Theorem 4.7 as a subroutine. In the latter parts, the algorithm tries out all possibilities to add vertices to a separator, however, we can show that we can upper-bound the number of possibilities in $\kappa + \ell$.

Theorem 4.8. *Given a temporal unit interval graph and vertex orderings that witness its shuffle number κ,* Temporal (s,z)-Separation *can be solved in $O((4\ell)^{\ell\cdot\kappa}\cdot(\kappa+\ell)\cdot|V|^2\cdot\ell^2)$ time.*

Proof. Let $\mathcal{G}=(V,(E_i)_{i\in[\ell]})$ be a temporal unit interval graph given as input together with vertex orderings $<_V^1,<_V^2,\ldots,<_V^\ell$, and let k be the size bound on the separator. The algorithm proceeds as follows. We introduce a set $M\subseteq V$ of "marked" vertices: We mark the terminals s and z as well as all vertices u,v with the property that for some $t\in[\ell-1]$ we have that $u<_V^t v$ and $v<_V^{t+1} u$, that is, their relative order is flipped at some point in time. More formally, let M be the largest subset of V that contains s and z with the property that for all $u\in M\setminus\{s,z\}$ there is a $v\in M$ and a $t\in[\ell-1]$ such that either $u<_V^t v$ and $v<_V^{t+1} u$, or $v<_V^t u$ and $u<_V^{t+1} v$.

Note that we can compute M in polynomial time using bubble sort when the vertex orderings are given. Furthermore, we have that $|M|\leq 2\cdot\kappa\cdot\ell+2$. If $M=V$, then we can solve the problem in the desired running time by trying out every possible separator. From now on we assume that $M\neq V$.

Next, we define two partitions, one for the vertex set M and one for the vertex set $V':=V\setminus M$. Intuitively, the partition of V' describes which parts of the orderings stay the same over the whole lifetime of the temporal graph, or in other words, which parts of the graph are order-preserving. The partition of M describes which vertices lie between the parts of the temporal graphs that are order-preserving. The partition is illustrated in Figure 4.5.

We define a partition of the vertices in $M=M_1\uplus M_2\uplus\ldots\uplus M_p$ as follows: Let $V=\{v_1,v_2,\ldots,v_n\}$ be the vertex ordering given by $<_V^1$ (that is, $v_i<_V^1 v_j$ if and only if $i<j$).

- We have that $s\in M_1$ and $z\in M_p$.
- If $v_i\in M$ and $v_{i+1}\in M$ for some $i\in[n-1]$, then $v_i\in M_j$ and $v_{i+1}\in M_j$ for some $j\in[p]$.
- If $v_i\in M_j$ and $v_{i'}\in M_j$ with $i<i'$ for some $j\in[p]$, then for all $i<i^\star<i'$ we have that $v_{i^\star}\in M_j$.
- For all $j\in[p]$ we have that $M_j\neq\emptyset$, and if v_i in M_j and $v_{i'}$ in M_{j+1} for some $j\in[p-1]$, then we have that $i<i'$.

Analogously, we define a partition of the remaining vertices $V'=V'_1\uplus V'_2\uplus\ldots\uplus V'_q$ in the following way:

- If $v_i\in V'$ and $v_{i+1}\in V'$ for some $i\in[n-1]$, then $v_i\in V'_j$ and $v_{i+1}\in V'_j$ for some $j\in[q]$.

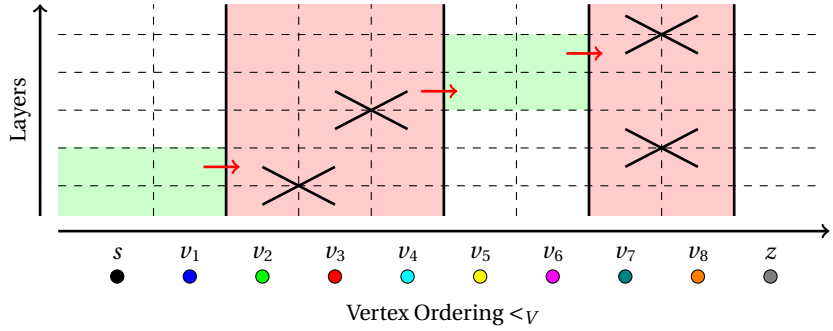


Figure 4.5: Visualization of the vertex partition created by the algorithm described in the proof of Theorem 4.8. Green regions illustrate parts of vertices that are order-preserving. Red regions illustrate parts of the vertices that are *not* order-preserving, indicated by the crossings. In the red regions, the algorithm "guesses" when a temporal path could enter and should leave the region, indicated by red arrows. These points in time then define subgraphs of the temporal unit interval graphs that are order-preserving (green regions) and are solved by the algorithm described in the proof of Theorem 4.7.

- If $v_i \in V'_j$ and $v_{i'} \in V'_j$ with $i < i'$ for some $j \in [q]$, then for all $i < i^\star < i'$ we have that $v_{i^\star} \in V'_j$.
- For all $j \in [q]$ we have that $V'_j \neq \emptyset$, and if v_i in V'_j and $v_{i'}$ in V'_{j+1} for some $j \in [q-1]$, then we have that $i < i'$.

We can easily compute both partitions by iterating over all vertices in V in the order given by $<^1_V$ and checking whether a vertex is contained in M. It is also easy to check that $q \leq p+1 \leq \kappa \cdot \ell + 3 \leq n$ since for all $1 < j < p$ we have that $|M_j| \geq 2$.

Note that any vertex ordering $<^t_V$ with $t \in [\ell]$ defines the same partitions.

Now we are ready to construct a separator S. First we guess the set $M_S := S \cap M$. Then for each $1 < j \leq p$ we guess the earliest time a_j a temporal path starting from s should be able to reach a vertex in the set M_j in $\mathcal{G}-S$ or we set $a_j := \ell+1$ if no temporal path from s should be able to reach a vertex in M_j in $\mathcal{G}-S$. For each $1 \leq j < p$ we guess the earliest time $d_j \geq a_j$ a temporal path from s should be able to reach a vertex in V'_j in $\mathcal{G}-S$ or, in other words, leave the set M_j, or we set $d_j := \ell+1$ if no temporal path from s should be able to reach a vertex in V'_j in $\mathcal{G}-S$.

Now we create the following instances of Temporal (s,z)-Separation on order-preserving temporal unit interval graphs: For each $j \in [q]$ we do the following: If $d_j < a_{j+1}$, then we create an order-preserving temporal unit interval graph by taking

the graph $\mathcal{G}|_{[d_j,a_{j+1}-1]}[V'_j]$ and adding two new vertices s_j and z_j. We further add the edge $\{s_j,u\}$ at time step t to the temporal graph if $d_j \leq t \leq a_{j+1}-1$ and $\{u',u\} \in E_t$ for some $u' \in M_j \setminus M_S$. We add the edge $\{z_j,u\}$ at time step t to the graph if $d_j \leq t \leq a_{j+1}-1$ and $\{u',u\} \in E_t$ for some $u' \in M_{j+1} \setminus M_S$. We call the constructed graph $\mathcal{G}_j$. Intuitively, we merge all vertices in $M_j \setminus M_S$ to a vertex s_j and all vertices in $M_{j+1} \setminus M_S$ to a vertex z_j. It is easy to check that $\mathcal{G}_j$ is an order-preserving temporal unit interval graph. Now we solve the optimization variant of Temporal (s,z)-Separation, where we want to minimize the separator size, on $(\mathcal{G}_j, s_j, z_j)$ using Theorem 4.7[9]. Let S_j be the solution, that is, a minimum temporal (s_j,z_j)-separator for $\mathcal{G}_j$. If there is no valid solution or if $d_j \geq a_{j+1}$, then we set $S_j = \emptyset$.

Finally, we set $S = \bigcup_{j\in[q]} S_j \cup M_S$. If $|S| \leq k$ and there is no temporal (s,z)-path in $\mathcal{G}-S$, then we output YES. Otherwise, we output NO.

It is easy to check that the algorithm runs in FPT-time with respect to parameter $(\kappa+\ell)$. We give a precise running time bound after the correctness proof.

Correctness. We next prove that the algorithm outputs YES if and only if we face a YES-instance.

($\Rightarrow$): If the algorithm outputs YES, then we face a YES-instance. This is trivially true since the algorithm does a sanity check as a last step.

($\Leftarrow$): If we face a YES-instance, then there is a temporal (s,z)-separator $S^\star$ with $|S^\star| \leq k$ for $\mathcal{G}$. We claim that in this case, our algorithm outputs YES. Since we try out all possible sets M_S we can assume that there is a branch of our algorithm where we have that $M_S = M \cap S^\star$. Similarly, we can assume that we are in a branch where all values a_j and d_j for $j \in [q]$ are "correct", that is, they are the largest numbers with the property that no vertex $v \in M_j$ is reachable from s in $\mathcal{G}-S^\star$ earlier than a_j and no vertex $u \in V'_j$ is reachable from s in $\mathcal{G}-S^\star$ earlier than d_j.

Then we can show that $S = \bigcup_{j\in[q]} S_j \cup M_S$ is a temporal (s,z)-separator and $|S| \leq |S^\star|$: We first check whether S is a temporal (s,z)-separator. Since $M \cap S^\star = M \cap S$, we know that for each part M_j with $1 < j < p$ we have that a temporal path from s that arrives at a vertex in M_j no earlier than a_j cannot arrive at a vertex in V'_j earlier than d_j in $\mathcal{G}-S$. Furthermore, no temporal path from s can arrive at a vertex in V'_1 earlier than d_1 in $\mathcal{G}-S$ and no temporal path from s that arrives at a vertex in M_p at time a_p or later can reach z in $\mathcal{G}-S$. The sets S_j are chosen in a way that ensures that a temporal path from s that does not arrive at any vertex in V'_j earlier than d_j cannot reach a vertex in M_{j+1} earlier than a_{j+1} in $\mathcal{G}-S_j$ and hence also in $\mathcal{G}-S$. We can conclude that S is a temporal (s,z)-separator for $\mathcal{G}$. Now assume for contradiction

[9] Note that the algorithm described in the proof of Theorem 4.7 can easily be modified to output a solution.

that $|S| > |S^\star|$. Then there is a set S_j such that $|S_j| > |V'_j \cap S^\star|$. This is a contradiction to the fact that S_j is a minimum temporal (s_j, z_j)-separator for $(\mathcal{G}_j, s_j, z_j)$ since $V'_j \cap S^\star$ is also a temporal (s_j, z_j)-separator for $(\mathcal{G}_j, s_j, z_j)$ since otherwise there would be a temporal path from s that arrives at a vertex in M_{j+1} earlier than a_{j+1} in $\mathcal{G} - S^\star$. This completes the correctness proof.

Running time. There are $2^{|M|}$ possible guesses for M_S and then a total of $\ell^{2(p-1)}$ possible guesses for the a_i-values and d_i-values. The polynomial part of the running time is $q \cdot O(|V|^2 \cdot \ell^2)$. Together with the bounds we have shown for q, p, and $|M|$ we get a running time upper bound of $O((4\ell)^{\ell \cdot \kappa} \cdot (\kappa + \ell) \cdot |V|^2 \cdot \ell^2)$. □

We leave as an open question how to effiently compute the shuffle number of a given temporal unit interval graph and a set of vertex orderings that witness the shuffle number. We conjecture that deciding whether a temporal unit interval graph has shuffle number $\kappa = 1$ is already NP-hard.

4.4 Restless Temporal Separators

In this section we investigate the computational complexity of Restless Temporal (s, z)-Separation. Since it generalizes Temporal (s, z)-Separation, we can quickly observe that the problem is NP-hard. Since the question whether a given temporal graph admits a Δ-restless temporal (s, z)-separator is the complement of the question whether the temporal graph contains a Δ-restless temporal (s, z)-path, we can also deduce that Restless Temporal (s, z)-Separation is coNP-hard since we know that Restless Temporal (s, z)-Path is NP-hard (Theorem 3.3). This already suggests that Restless Temporal (s, z)-Separation is located somewhere higher in the polynomial time hierarchy. Indeed we can show that Restless Temporal (s, z)-Separation is Σ_2^{P}-complete. This implies, for example, that we presumably cannot use SAT-solvers or ILP-solvers to compute Δ-restless temporal (s, z)-separators. To show Σ_2^{P}-hardness, we give a reduction from $\exists\forall$-SAT, where we are given a Boolean formula ϕ in conjunctive normal form and a partition of variables of ϕ into two sets X and Y. Then we are asked to decide whether there exists an assignment for all variables in X such that for all possible assignments for the variables in Y, the formula ϕ evaluates to true. The very rough idea of our reduction is that the vertices selected for the separator correspond to an assignment for the variables in X and if there is an assignment for the variables in Y such that ϕ evaluates to false, then the temporal graph should still contain a Δ-restless temporal (s, z)-path after the separator vertices are removed.

Theorem 4.9. RESTLESS TEMPORAL (s,z)-SEPARATION *is* Σ_2^P*-complete for all* $\Delta \geq 1$ *even if every edge has only one time stamp.*

Proof. We present a polynomial-time reduction from the Σ_2^P-complete problem $\exists\forall$-SAT [AB09, Sto76], where we are given a Boolean formula ϕ in conjunctive normal form and the variables of ϕ are partitioned into two sets X and Y, and we are asked to decide whether there exists an assignment for all variables in X such that for all possible assignments for the variables in Y, the formula ϕ evaluates to true.

Let $\phi(X,Y)$ denote an instance of $\exists\forall$-SAT, let $n_X = |X|$, $n_Y = |Y|$, and let m be the number of clauses in ϕ. We assume that no clause of ϕ contains a variable several times. We construct a temporal graph $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$ with $\ell = 2m+1$, consisting of three gadgets. We start with the "exists gadget" in which we have to select the vertices of the Δ-restless temporal (s,z)-separator. Intuitively, this chooses an assignment for the variables in X. The next gadget is the "forall gadget" which must be passed by every Δ-restless temporal (s,z)-path. This gadget can be traversed in 2^{n_Y} ways which, intuitively, represent all possible assignments for the variables in Y. The last gadget is the clause gadget which, intuitively, a Δ-restless temporal (s,z)-path can only pass if there is a clause that is not satisfied. We set $\Delta = 1$ and $k = n_X$. We next give formal descriptions of the gadgets.

Exists Gadget. We start by creating two vertices s and z. For every variable $x_i \in X$ we create two vertices $x_i^{(T)}$ and $x_i^{(F)}$ and we add edges $\{s,x_i^{(T)}\}$, $\{x_i^{(T)},x_i^{(F)}\}$, and $\{x_i^{(F)},z\}$ to E_1. This already completes the construction of the exists gadget. We can see that we created n_X internally vertex-disjoint Δ-restless temporal (s,z)-paths. Since we set $k = n_X$, we have that every Δ-restless temporal (s,z)-separator has to contain one vertex from each of these paths.

Forall Gadget. For every variable $y_i \in Y$ we create two vertices $y_i^{(T)}$ and $y_i^{(F)}$. We further create n_Y+1 vertices $s_1,s_2,\dots s_{n_Y+1}$. For all $i \in [n_Y]$ we add edges $\{s_i,y_i^{(T)}\}$, $\{s_i,y_i^{(F)}\}$, $\{y_i^{(T)},s_{i+1}\}$, and $\{y_i^{(F)},s_{i+1}\}$ to E_1. We further add edge $\{s,s_1\}$ to E_1. This completes the construction of the forall gadget. We can see that there are 2^{n_Y} different Δ-restless temporal paths from s_1 to s_{n_Y+1}. Intuitively, each one of these represents an assignment for the variables in Y.

Clause Gadget. For every clause c_i of ϕ we create two vertices $c_i^{(1)}$ and $c_i^{(2)}$. For every $i \in [m]$ we add edge $\{c_i^{(1)},c_i^{(2)}\}$ to E_{2i+1} and if $i < m$, then we add edge $\{c_i^{(2)},c_{i+1}^{(1)}\}$ to E_{2i+2}. We further add edge $\{s_{n_Y+1},c_1^{(1)}\}$ to E_2. We call this part of the gadget the *clause selection path.*

Let c_i be a clause for some $i \in [m]$. Without loss of generality let c_i contain variables $x_1,\dots,x_{j_1}$ and $y_1,\dots,y_{j_2}$. Then we add the following edges to E_{2i+1}:

- If x_1 appears non-negated in c_i, then we add edge $\{c_i^{(2)}, x_1^{(F)}\}$, otherwise we add edge $\{c_i^{(2)}, x_1^{(T)}\}$.
- For all $j \in [j_1 - 1]$, if x_j appears non-negated in c_i, then set $v_j = x_j^{(F)}$, otherwise set $v_j = x_j^{(T)}$. If x_{j+1} appears non-negated in c_i, then set $v_{j+1} = x_{j+1}^{(F)}$, otherwise set $v_{j+1} = x_{j+1}^{(T)}$. We add edge $\{v_j, v_{j+1}\}$.
- If x_{j_1} appears non-negated in c_i, then set $v = x_{j_1}^{(F)}$, otherwise set $v = x_{j_1}^{(T)}$. If y_1 appears non-negated in c_i, then set $w = y_1^{(F)}$, otherwise set $w = y_1^{(T)}$. We add edge $\{v, w\}$.
- For all $j \in [j_2 - 1]$, if y_j appears non-negated in c_i, then set $v_j = y_j^{(F)}$, otherwise set $v_j = y_j^{(T)}$. If y_{j+1} appears non-negated in c_i, then set $v_{j+1} = y_{j+1}^{(F)}$, otherwise set $v_{j+1} = y_{j+1}^{(T)}$. We add edge $\{v_j, v_{j+1}\}$.
- If y_{j_2} appears non-negated in c_i, then we add edge $\{y_{j_2}^{(F)}, z\}$, otherwise we add edge $\{y_{j_2}^{(T)}, z\}$.

We do this for all clauses in ϕ. This completes the construction of the clause gadget. Intuitively, a Δ-restless temporal (s,z)-path should only be able to traverse the clause gadget if there is a clause that is not satisfied.

This finishes the construction of $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$. The construction is illustrated in Figure 4.6. Recall that $\Delta = 1$ and $k = n_X$. It is easy to check that $\mathcal{G}$ can be constructed in polynomial time and that every edge has at most one time stamp.

Correctness. Now we show that $\mathcal{G}$ admits a Δ-restless temporal (s,z)-separator of size at most k if and only if ϕ is a YES-instance of $\exists\forall$-SAT.

($\Rightarrow$): Assume that there is an assignment for the variables in X such that for all assignments for the variables of Y we have that ϕ evaluates to true. We construct a Δ-restless temporal (s,z)-separator S for $\mathcal{G}$ as follows. For each $i \in [n_X]$, if variable x_i is assigned the value true, then we add the vertex $x_i^{(T)}$ to S, otherwise we add $x_i^{(F)}$ to S. Clearly, we have that $|S| = n_X = k$. In the following, we show that S is a Δ-restless temporal (s,z)-separator for $\mathcal{G}$.

Assume for contradiction that there is a Δ-restless temporal (s,z)-path P in $\mathcal{G} - S$. It is easy to see that all Δ-restless temporal (s,z)-paths in $\mathcal{G}$ that only use edges from the exists gadget are destroyed in $\mathcal{G} - S$ since from every such path, we put one vertex into S. Observe that all time edges adjacent to z that are not part of the exists gadget have a time stamp of three or larger. Hence, to reach a time edge with time step two, P has to pass the forall gadget to reach time edge $\{s_{n_Y+1}, c_1^{(1)}\}$, which is the only

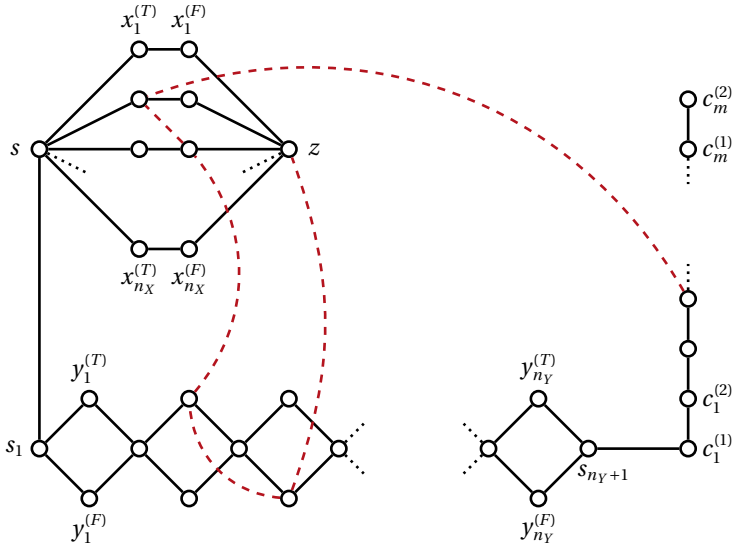


Figure 4.6: Visualization of parts of the underlying graph of the temporal graph $\mathscr{G}$ constructed in the reduction of Theorem 4.9. The red dashed path corresponds to the clause gadget for clause $c_2 = (\neg x_2 \vee x_3 \vee \neg y_2 \vee y_3)$ (without the clause selection path).

time edge with time stamp two. From this it follows that for every $i \in [n_Y]$ we have that either $y_i^{(T)} \in V(P)$ or $y_i^{(F)} \in V(P)$. Then the path P enters the clause selection path of the clause gadget. To reach z, the path P has to leave this path at some vertex $c_j^{(2)}$ for some $j \in [m]$ (meaning that $c_j^{(2)} \in V(P)$ and $c_{j+1}^{(1)} \notin V(P)$). We claim that this implies that clause c_j is not satisfied if the variables from Y are assigned the following truth values: for each $i \in [n_Y]$, if $y_i^{(T)} \in V(P)$, then we set y_i to true, otherwise we set y_i to false. Assuming that $c_j^{(2)} \in V(P)$ and $c_{j+1}^{(1)} \notin V(P)$, the only way to reach z from $c_j^{(2)}$ is through vertices that correspond to the variables appearing in clause c_i, since the time stamps from all paths from the clause selection path to z differ by at least two. More specifically, for each variable x_i (y_i) appearing in c_j, we have that $V(P)$ contains vertex $x_i^{(F)}$ ($y_i^{(F)}$) if x_i (y_i) appears non-negated in c_j and $V(P)$ contains vertex $x_i^{(T)}$ ($y_i^{(T)}$) otherwise. This means for the variables x_i that they are set to truth values that do not satisfy clause c_j, otherwise the corresponding vertices would be contained in the separator S. For the variables y_i this means the assignment we constructed earlier also sets them to truth values that do not satisfy clause c_j,

otherwise the corresponding vertices would have been used by P when the path was passing the forall gadget at time step one. Hence, we have found an assignment for the variables in Y such that together with the given assignment for the variables in X, the formula ϕ evaluates to false—a contradiction.

($\Leftarrow$): Let $S \subseteq V \setminus \{s,z\}$ with $|S| \leq k$ be a Δ-restless temporal (s,z)-separator for $\mathscr{G}$. Let us first look at the exists gadget of $\mathscr{G}$. It consists of n_X internally vertex-disjoint Δ-restless temporal (s,z)-paths, each one visiting four vertices: s, $x_i^{(T)}$, $x_i^{(F)}$, and z for some $i \in [n_X]$. Of each of these Δ-restless temporal (s,z)-paths, one vertex other than s or z has to be contained in S, otherwise S would not be a Δ-restless temporal (s,z)-separator. It follows that for all $i \in [n_X]$ either $x_i^{(T)}$ or $x_i^{(F)}$ is contained in S (and also no other vertices are contained in S since $k = n_X$). This lets us construct an assignment for the variables in X as follows. For every $i \in [n_X]$, if $x_i^{(T)} \in S$, then we set x_i to true, otherwise we set x_i to false. We claim that using this assignment for the variables in X, we have that for all assignments for the variables in Y the formula ϕ evaluates to true.

Assume for the sake of contradiction that there is an assignment for the variables in Y such that together with the constructed assignment for the variables in X, the formula ϕ evaluates to false. Then we can construct a Δ-restless temporal (s,z)-path in $\mathscr{G} - S$ as follows. Starting from s we traverse the forall gadget as follows. Starting with $i = 1$ to n_Y we visit s_i and then $y_i^{(T)}$ if y_i is set to true, and $y_i^{(F)}$ otherwise. Then we visit s_{n_Y+1}. Up until this point, the path only uses time edges with time stamp one. Since ϕ evaluates to false, there is at least one clause in ϕ that is not satisfied. Let c_j be that clause. We continue our Δ-restless temporal path from s_{n_Y+1} to $c_j^{(2)}$. Since c_j evaluates to false, the vertices corresponding to the variables in X appearing in c_j are not contained in S, otherwise, by construction, the clause c_j would evaluate to true. Similarly, the vertices corresponding to the variables in Y appearing in c_j have not been visited by the path when traversing the forall gadget. Hence, we can continue the Δ-restless temporal path from $c_j^{(2)}$ to z—a contradiction.

Containment in Σ_2^P. Our proof so far shows that Restless Temporal (s,z)-Separation is Σ_2^{P}-hard. To show that the problem is Σ_2^{P}-complete, we show that it is contained in Σ_2^{P}. Recall that Σ_2^{P} contains all problems that can be solved by an NP-machine that has oracle access to an NP-complete problem [AB09, Sto76]. We can solve an instance $\mathscr{G} = (V, (E_i)_{i\in[\ell]}), s, z, k, \Delta)$ of Restless Temporal (s,z)-Separation with such a machine as follows. We non-deterministically guess a set $S \subseteq V$ of size at most k and then produce an instance $(\mathscr{G} - S, s, z, \Delta)$ of Restless Temporal (s,z)-Path. Since Restless Temporal (s,z)-Path is contained in NP, we can reduce it to the NP-complete problem to which we have oracle access. We use the reduction to

produce an equivalent instance of the NP-complete problem we have oracle access to and query the oracle with this instance. If the oracle answers NO, then we have found a Δ-restless temporal (s,z)-separator of size at most k for $\mathcal{G}$ and can answer YES. It is easy to see that the described machine has an accepting path if and only if the RESTLESS TEMPORAL (s,z)-SEPARATION instance is a YES-instance. □

From a parameterized complexity perspective we can make one rather straightforward observation. Since RESTLESS TEMPORAL (s,z)-SEPARATION generalizes TEMPORAL (s,z)-SEPARATION, we know that RESTLESS TEMPORAL (s,z)-SEPARATION parameterized by the separator size k is W[1]-hard [Zsc+20]. However, we can observe that RESTLESS TEMPORAL (s,z)-SEPARATION is even W[2]-hard when parameterized by the separator size k by a straightforward reduction from HITTING SET, where we model each element of the universe with a vertex and each set by a path through the corresponding vertices. The waiting time Δ allows us to obtain a one-to-one correspondence between restless temporal paths in the constructed temporal graph and sets in the HITTING SET instance. We remark that the reduction we use to show this result has been used in a very similar way by Zschoche [Zsc17] to show that finding temporal separators of bounded size that destroy all Δ-restless temporal *walks* from s to z is W[2]-hard when parameterized by the bound on the separator size.

Observation 4.10. RESTLESS TEMPORAL (s,z)-SEPARATION *parameterized by the separator size k is W[2]-hard for all* $\Delta \geq 1$.

Proof. We present a parameterized polynomial-time reduction from HITTING SET, where we are given a universe set U, a family of sets $S_1, \ldots, S_m \subseteq U$, and an integer h, and are asked whether there is a *hitting set* $S^\star \subseteq U$ with $|S^\star| \leq h$ such that for all $i \in [m]$ we have that $S^\star \cap S_i \neq \emptyset$. HITTING SET is W[2]-complete when parameterized by h [DF99, PM81].

Given an instance $(U, (S_i)_{i\in[m]}, h)$ of HITTING SET, we construct a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ with $\ell = 2m$ as follows. We set $V = U \cup \{s,z\}$ and for each set S_i with $i \in [m]$ we create two layers G_{2i-1} and G_{2i}. In layer G_{2i-1} we create a path from s to z that visits all vertices in S_i in an arbitrary order. The layer G_{2i} is trivial. We set $\Delta = 1$ and $k = h$. This finishes the construction. It is easy to check that this can be done in polynomial time.

Correctness. The correctness is straightforward to see. A Δ-restless temporal (s,z)-separator for $\mathcal{G}$ has to contain at least one vertex from each set S_i with $i \in [m]$, otherwise there would be a layer that contains a Δ-restless temporal (s,z)-path. It follows that a Δ-restless temporal (s,z)-separator for $\mathcal{G}$ is a hitting set for $(U, (S_i)_{i\in[m]})$.

For the other direction, one has to observe that due to the waiting time restriction Δ and the trivial layers that are present in $\mathcal{G}$, each Δ-restless temporal (s,z)-path in $\mathcal{G}$ corresponds to a set S_i for some $i \in [m]$ of the HITTING SET instance. It follows that a hitting set contains at least one vertex from each Δ-restless temporal (s,z)-path in $\mathcal{G}$. □

We remark that it is open whether TEMPORAL (s,z)-SEPARATION parameterized by the separator size k is contained in W[1]. Hence, Observation 4.10 does not necessarily imply that RESTLESS TEMPORAL (s,z)-SEPARATION parameterized by the separator size k is harder than TEMPORAL (s,z)-SEPARATION parameterized by the separator size k. Also containment of RESTLESS TEMPORAL (s,z)-SEPARATION parameterized by the separator size k in W[2] is open.

4.5 Conclusion

In the first part of this chapter we investigated the computational complexity of finding temporal separators on temporal unit interval graphs, a restricted class of temporal graphs that is motivated from physical proximity network modeling. This approach allowed us to obtain fixed-parameter tractability results for a novel type of parameter, the "shuffle number", that measures how drastically temporal unit interval graphs change over time. These results are a first step towards better understanding temporal graphs based on geometric intersection models. Future work includes generalizing these results to, for example, temporal graphs based on square or disk intersection models.

Furthermore, we believe that these restrictions are well motivated for several other problems on temporal graphs, such as for example TEMPORAL CLIQUE, a problem that has been studied in the context of physical proximity networks [Ben+19, Him+17, VLM16] and which we investigate (on general temporal graphs) in Chapter 7.

In the second part of this chapter, we adopted the path model we studied in Chapter 3 for temporal separators and investigated the computational complexity of finding *restless* temporal separators. We established that this problem is hard for Σ_2^P, a complexity class that is located in the second level of the polynomial time hierarchy. This implies, for example, that we presumably cannot use SAT-solvers or ILP-solvers to compute restless temporal separators. So far, we do not know any tractability results for this problem yet and leave that as a task for future research.

CHAPTER 5

Temporal Matchings

Finding matchings is one of the most important primitives in graph algorithmics. In this chapter, we study the computational complexity of finding matchings in temporal graphs. Our model of temporal matching (which appears to be slightly more general than a previous one due to Baste, Bui-Xuan, and Roux [BBR20]) allows for capturing several real-world scenarios where (as for example in social networks) relations change over time and where one also has to model the duration of pairings between agents. Intuitively, we say that if two agents are matched at some time step, then they both cannot be matched again in the next Δ time steps. This is motivated from modeling activities that require a "recovery time" after completion. We present several computational hardness results for very restricted cases and employ *temporal line graphs* as an important tool. In particular, we show that computing matchings of a given minimum cardinality is NP-hard even if the underlying graph of the input temporal graph is a path. This presumably excludes fixed-parameter tractability results for several popular and useful parameterizations.

This chapter is based on the paper "Computing maximum matchings in temporal graphs" by Mertzios et al. [Mer+20].

5.1 Introduction

Computing maximum matchings (maximum-cardinality sets of independent edges in an undirected graph) is one of the most fundamental graph-algorithmic primitives [LP09]. In this work, we lift the study of the algorithmic complexity of finding maximum matchings from static graphs to temporal graphs. Whenever facing network structures changing over time (for example, social or biological networks), the search for "temporal matchings" is a fundamental task. To this end, however, one first has to come up with a natural model of temporal matching which, indeed, leaves quite some degrees of freedom. We address this next.

We investigate a model for temporal matchings that is inspired by the work of Baste, Bui-Xuan, and Roux [BBR20]. We build on their work and answer some of their open questions. Our model slightly differs from theirs. Indeed, we have an easy reduction from their model to ours whereas it is not clear whether an equally easy reduction also exists for the opposite direction. In our model, we search for

Δ-*temporal matchings*. Roughly speaking, two time-stamped edges are *temporally independent* (with respect to a natural number Δ) if their unlabeled versions do not share an endpoint or their time stamps differ by at least Δ. While we search for such temporally independent edges, Baste, Bui-Xuan, and Roux [BBR20] additionally request that, in order to be eligible for a matching, an edge must exist in the input in at least Δ consecutive layers. Thus, their matchings need to consist of "time-consecutive edge blocks", which requires some "data cleaning" to make their model fit with real-world "link stream" data in their experiments [BBR20].

To the best of our knowledge, the main alternative model for temporal matchings in temporal graphs is the concept of multistage (perfect) matchings [Bam+18, GTW14]. This model, which is inspired by reconfiguration or reoptimization problems, is not directly related to ours. Roughly speaking, the goal is to find perfect matchings for every layer of a temporal graph such that the matchings only change slowly over time.

Before proceeding with a more general discussion of related work and our results, let us briefly discuss a motivating example for finding Δ-temporal matchings. Assume that each vertex represents a police(wo)man. An edge labeled with a number t then means that the two corresponding persons are available to perform a joint activity at this time step t. In our police setting, this could mean to be together on patrol on a specific day t. With the time window length Δ we model the length of the "recovery time" that is required after the activity. In the police setting this could mean that the two police(wo)men cannot (or do not need to) patrol for Δ days. More generally, once two entities (vertices) participate in an activity (time-labeled edge) at some time step t, at least Δ time steps (the recovery time) need to pass after t before any of these entities (vertices) can again become available for any other activity.

5.1.1 Related Work

The work of Baste, Bui-Xuan, and Roux [BBR20] is by far the closest to the setting we study in this chapter. Among other things, they showed that finding maximum matchings in temporal graphs is NP-hard, even when $\Delta \geq 2$. In terms of parameterized complexity, they provided a polynomial-size problem kernel for the combined parameter $k + \Delta$, where k is the lower bound for the cardinality of the wanted matching. Finally, they presented a polynomial-time 1/2-approximation algorithm and conducted some experimental work [BBR20]. We mention in passing that all their algorithmic results (both positive and negative) easily translate to our setting.

Gupta, Talwar, and Wieder [GTW14] introduced the concept of multi-stage (perfect) matchings. Here, the goal is to find perfect matchings for every layer of a temporal graph such that the (symmetric) difference of the matchings of two adjacent

layers is small. In this setting one mostly encounters computational intractability, which leads to several results on approximation hardness and algorithms [Bam+18, GTW14]. Furthermore, we remark that Michail and Spirakis [MS16] defined a different specialized notion of temporal matching which they used as an auxiliary tool to prove computational hardness results for traveling salesperson problems in temporal graphs.

In the closely related *multi-layer* setting, generalizations of matchings have also been studied. Recall that in contrast to temporal graphs which are ordered sequences of layers, a multi-layered graph is an *unordered* set of layers. For multi-layered graphs with two layers it has been shown that a perfect matching can be computed in polynomial time, while three layers already yield NP-hardness [Bre+19].

Notably, in static graphs there is a close connection between finding matchings and finding vertex covers (that is, sets of vertices that cover all edges). Very recently, finding vertex covers in temporal graphs has been studied in various models [Akr+20, Flu+19]. However, we could not observe any direct links to be exploited between vertex covering and matching in the temporal setting.

Finding maximum matchings in static graphs is a classic polynomial-time solvable graph problem [MV80] that has been studies extensively. For an overview on results for matchings in static graphs, we refer to the book of Lovász and Plummer [LP09].

5.1.2 Our Contributions and Organization of the Chapter

We introduce the TEMPORAL MATCHING problem, thoroughly investigate its computational hardness and thereby show that known algorithms presumably cannot be improved significantly. First, we prove that it is NP-complete even if the input temporal graph has only three layers. is complete.

The second major result of this chapter is to show that TEMPORAL MATCHING remains NP-hard even in the very restricted case where the underlying graph of the temporal input graph is a path. On our way to prove the above hardness results, we make use the notion of a *temporal line graph*. To the best of our knowledge this is the first non-trivial application of this concept which may prove useful in other contexts, too. Temporal line graphs form a restricted class of *static* graphs, which remains largely unexplored. This notion enables us to reduce the problem of computing a large temporal matching to the problem of computing a large independent set in a static graph (namely in the temporal line graph that is specified by the input temporal graph). Moreover, as an intermediate result, we show that the classic problem INDEPENDENT SET (on static graphs) remains NP-hard on induced subgraphs of *diagonal grid graphs*, thus strengthening an old result of Clark, Colbourn, and Johnson [CCJ90] for unit disk graphs.

The chapter is organized as follows. In Section 5.2, we formally define Δ-temporal matchings, our main problem TEMPORAL MATCHING, and temporal line graphs. We further present some preliminary computational complexity observations. Moreover, we discuss the relationship between our model for temporal matchings and the one of Baste, Bui-Xuan, and Roux [BBR20] in more detail. We continue by presenting our first NP-hardness reduction in Section 5.3 and prove in Section 5.4 that TEMPORAL MATCHING is NP-hard even if the underlying graph of the input temporal graph is a path. Here, we make extensive use of the concept of temporal line graphs. We conclude in Section 5.5.

5.1.3 Further Contributions of the Paper this Chapter is Based on

Additionally to the contributions we present in this chapter, Mertzios et al. [Mer+20] show that the optimization variant of TEMPORAL MATCHING where the size of the temporal matching should be maximized is APX-complete. On the positive side, they provide a polynomial-time $\Delta/(2\Delta-1)$-approximation algorithm. Furthermore, they show that TEMPORAL MATCHING is fixed-parameter tractable with respect to the cardinality k of the sought temporal matching and with respect to the combination of the vertex cover number of the underlying graph and the time window size Δ.

5.2 Preliminaries

In this section, we provide additional definitions and notation related to matchings and line graphs. In particular, we present the formal definitions of temporal matchings, temporal line graphs, and give the formal problem definition of TEMPORAL MATCHING and some preliminary computational complexity observations. We also briefly discuss the relation between our model of temporal matchings and the model used by Baste, Bui-Xuan, and Roux [BBR20].

5.2.1 Temporal Matchings

A *matching* in a (static) graph $G = (V, E)$ is a set $M \subseteq E$ of edges such that for all $e, e' \in M$ we have that $e \cap e' = \emptyset$ [LP09]. In the following, we transfer this concept to temporal graphs.

For a natural number Δ, two time-edges (e, t), (e', t') are Δ-*independent* if $e \cap e' = \emptyset$ or $|t - t'| \geq \Delta$. If two time-edges are not Δ-independent, then we say that they are *in conflict*. A time-edge (e, t) Δ-*blocks* a vertex appearance (v, t') (or (v, t') is Δ-*blocked* by (e, t)) if $v \in e$ and $|t - t'| \leq \Delta - 1$. A Δ-*temporal matching* M of a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ is a set of time-edges of $\mathcal{G}$ which are pairwise Δ-independent. Formally, it is defined as follows.

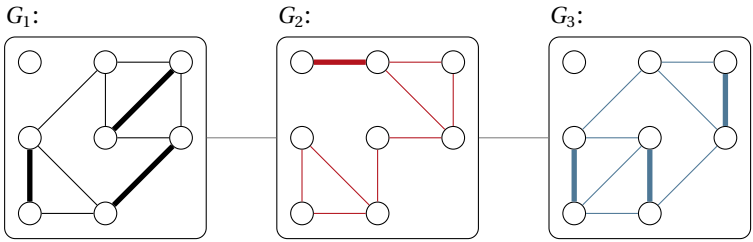


Figure 5.1: Example temporal graph with lifetime three. The thick edges form a maximum 2-temporal matching.

Definition 5.1 (Δ-Temporal Matching)**.** A Δ-*temporal matching* of a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ is a set $M \subseteq E(G_|) \times [\ell]$ of time-edges of $\mathcal{G}$ such that for all $(e,t) \in M$ we have that $e \in E_t$ and for every pair of distinct time-edges $(e,t),(e',t')$ in M we have that $e \cap e' = \emptyset$ or $|t - t'| \geq \Delta$.

We remark that this definition is similar to the definition of γ-*matchings* by Baste, Bui-Xuan, and Roux [BBR20]. We point out similarities and differences in a dedicated paragraph at the end of this section.

A Δ-temporal matching is called *maximal* if it is not properly contained in any other Δ-temporal matching. A Δ-temporal matching is called *maximum* if there is no Δ-temporal matching of larger cardinality. For an example of a Δ-temporal matching see Figure 5.1.

Having defined temporal matchings, we naturally arrive at the following central problem.

TEMPORAL MATCHING	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.
Question:	Is there a Δ-temporal matching in $\mathcal{G}$ of cardinality at least k?

It is easy to see that we can check in polynomial time whether, given a temporal graph, a given set of time-edges is a Δ-temporal matching. This implies that TEMPORAL MATCHING is contained in NP and in subsequent NP-completeness statements we will only discuss hardness.

5.2.2 Temporal Line Graphs

The *line graph* of a (static) graph $G = (V,E)$ is a graph $L(G)$ with vertex set $V(L(G)) = \{v_e \mid e \in E\}$ and for all $v_e, v_{e'} \in V(L(G))$ we have that $\{v_e, v_{e'}\} \in E(L(G))$ if and only

if $e \cap e' \neq \emptyset$ [Die16]. In the context of matchings, line graphs are of special interest since the cardinality of a maximum matching in a graph equals the cardinality of a maximum independent set in its line graph. Indeed, a matching in a graph can directly be translated into an independent set in its line graph and vice versa [Die16]. In the following, we show how to transfer this concept to temporal graphs and temporal matchings. In particular, we make use of temporal line graphs in the NP-hardness result of Section 5.4.

The Δ-*temporal line graph* of a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ is a static graph that has a vertex for every time-edge of $\mathcal{G}$ and two vertices are connected by an edge if the corresponding time-edges are in conflict, that is, they cannot be both part of a Δ-temporal matching of $\mathcal{G}$. We say that a graph H is a *temporal line graph* if there exists a Δ and a temporal graph $\mathcal{G}$ such that H is isomorphic to the Δ-temporal line graph of $\mathcal{G}$. Formally, temporal line graphs and Δ-temporal line graphs are defined as follows.

Definition 5.2 (Temporal Line Graph)**.** Given a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ and an integer $\Delta \in \mathbb{N}$, the Δ-*temporal line graph* $L_\Delta(\mathcal{G})$ of $\mathcal{G}$ is defined as follows.

- $V(L_\Delta(\mathcal{G})) := \{e_t \mid e \in E_t\}$,
- $E(L_\Delta(\mathcal{G})) := \{\{e_t, e'_{t'}\} \mid e \cap e' \neq \emptyset \wedge |t - t'| < \Delta\}$.

We say that a graph H is a *temporal line graph* if there is a temporal graph $\mathcal{G}$ and an integer Δ such that $H = L_\Delta(\mathcal{G})$.

By definition, Δ-temporal line graphs have the following property.

Observation 5.1. *Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a temporal graph and let $L_\Delta(\mathcal{G})$ be its Δ-temporal line graph. The cardinality of a maximum independent set in $L_\Delta(\mathcal{G})$ equals the size of a maximum Δ-temporal matching of $\mathcal{G}$.*

It follows that solving Temporal Matching on a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ is equivalent to solving Independent Set on $L_\Delta(\mathcal{G})$.

We remark that a different notion of temporal line graphs was introduced in a survey by Latapy, Viard, and Magnien [LVM18], which is somewhat similar to our definition for the case of $\Delta = 1$.

5.2.3 Basic Observations

Note that when the input parameter Δ in Temporal Matching is equal to one, the problem can be solved efficiently, because it reduces to ℓ independent instances of (static) Maximum Matching, which can be solved in polynomial time [MV80].

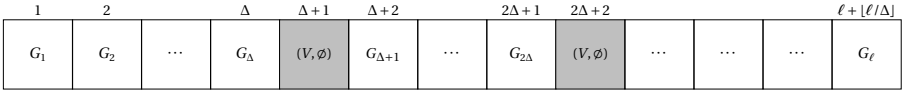


Figure 5.2: Inserting trivial layers to reduce TEMPORAL MATCHING on instances $(\mathcal{G}, \Delta, k)$ to TEMPORAL MATCHING on instances $(\mathcal{G}, \Delta+1, k)$.

At the other extreme are instances $(\mathcal{G} = (V, (E_i)_{i\in[\ell]}), \Delta, k)$ in which Δ coincides with the lifetime ℓ, that is, $\Delta = \ell$. In this case the problem can also be solved in polynomial time. Indeed, it can again be reduced to solving MAXIMUM MATCHING, this time in the underlying graph $G_\downarrow$. Every matching edge in the underlying graph is then put into the temporal matching with some arbitrary time step where this edge is present.

Furthermore, it is easy to observe that computational hardness of TEMPORAL MATCHING for some fixed value of Δ implies hardness for all larger values of Δ. This allows us to construct hardness reductions for small fixed values of Δ and still obtain general hardness results.

Observation 5.2. *For every fixed* Δ, TEMPORAL MATCHING *on instances* $(\mathcal{G}, \Delta+1, k)$ *is computationally at least as hard as* TEMPORAL MATCHING *on instances* $(\mathcal{G}, \Delta, k)$.

Proof. The claim immediately follows from the observation that a temporal graph $\mathcal{G}$ contains a Δ-temporal matching of size at least k if and only if the temporal graph $\mathcal{G}'$ contains a $(\Delta+1)$-temporal matching of size at least k, where $\mathcal{G}'$ is obtained from $\mathcal{G}$ by inserting one trivial layer after every Δ consecutive layers (see Figure 5.2). □

Next, we briefly discuss some further results from Mertzios et al. [Mer+20] that are not part of this chapter. Mertzios et al. [Mer+20, Theorem 22] showed that TEMPORAL MATCHING admits a factor-$\frac{\Delta}{2\Delta-1}$ approximation algorithm. They further showed that TEMPORAL MATCHING admits an FPT-algorithm for the parameter k (matching size) [Mer+20, Theorem 25] and one for the combined parameter $\Delta + \mathrm{vc}_\downarrow$ [Mer+20, Theorem 34], where $\mathrm{vc}_\downarrow$ denotes the vertex cover number of the underlying graph.

5.2.4 Relation to γ-MATCHING by Baste, Bui-Xuan, and Roux [BBR20]

We refer to the variant of temporal matching introduced by Baste, Bui-Xuan, and Roux [BBR20] as γ-MATCHING. They defined the problem γ-MATCHING very similarly to the way we define TEMPORAL MATCHING. Their definition requires a time-edge to be present at γ consecutive time steps to be eligible for a temporal matching. There is an easy reduction from their model to ours: For every sequence of γ consecutive time-edges starting at time step t, we introduce *just one* time-edge at time step t, and

set Δ to γ. This already implies that Temporal Matching is NP-complete [BBR20, Theorem 1] and that algorithmic results for Temporal Matching also hold for γ-Matching. We do not know an equally easy reduction in the reverse direction.

In addition, it is easy to check that the algorithmic results of Baste, Bui-Xuan, and Roux [BBR20] can also be adapted to our model. Hence, we get that Temporal Matching admits a polynomial kernel when parameterized by $k+\Delta$ [BBR20, Theorem 2]. Some of our hardness results can also easily be transferred to γ-Matching. Whenever this is the case, we indicate this.

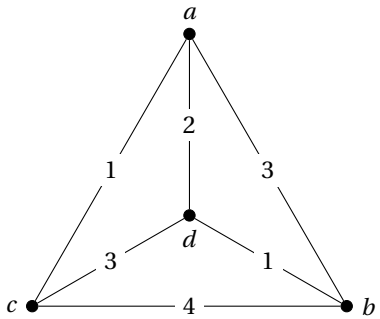
5.3 NP-Hardness of Temporal Matching with Few Layers

In this section, we prove that Temporal Matching is NP-complete even if the input temporal graph has only a constant number of layers. This is an improvement over the previously known NP-hardness results by Baste, Bui-Xuan, and Roux [BBR20] since their reduction has an unbounded number of layers.

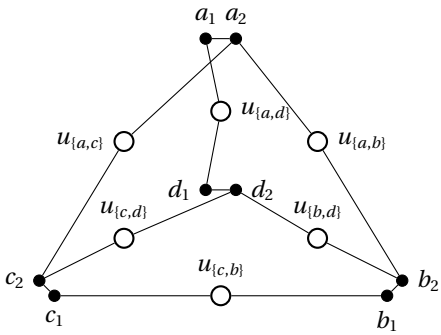
Theorem 5.3. Temporal Matching *is NP-complete for all* $\Delta \geq 2$ *and* $\ell \geq \Delta + 1$ *even if the underlying graph has maximum degree three and every edge of the underlying graph appears only once.*

Proof. We present a polynomial-time reduction from Independent Set, which is known to be NP-complete even on graphs with maximum degree three [GJ79, GJS76]. In this problem we are asked to decide whether a given graph $H = (U, F)$ with maximum degree three contains a set of at least h pairwise non-adjacent vertices. Let $H = (U, F)$ be a graph with maximum degree three and let h be an integer. We construct in polynomial time a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ with lifetime $\ell = 3$ as follows (to obtain instances with larger lifetime, we can simply add additional trivial layers after the first layer). First, we find a proper 4-edge coloring $\Upsilon : E \to \{1,2,3,4\}$ of H. Such a coloring exists by Vizing's theorem [Viz64] and can be found in $O(|F|)$ time [Sch98]. For every vertex $v \in U$ of H we add two vertices v_1, v_2 to the vertex set V of $\mathcal{G}$. For every edge $e \in F$ of H we add a vertex u_e to V. Next, for every vertex $v \in U$ we add the following edges to $\mathcal{G}$.

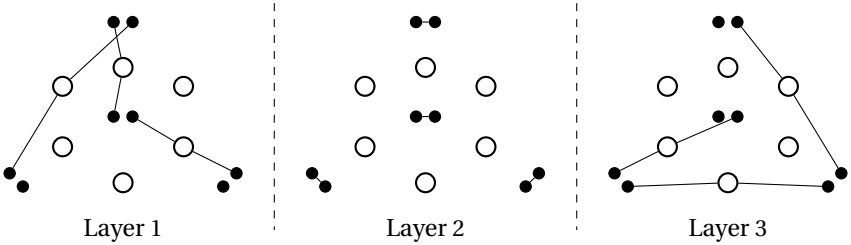
1. We add $\{v_1, v_2\}$ to E_2.
2. For every edge $e = \{v, w\} \in F$ such that $\Upsilon(e) \in \{1,2\}$ we add edges $\{u_e, v_{\Upsilon(e)}\}$ and $\{u_e, w_{\Upsilon(e)}\}$ to E_1.
3. For every edge $e = \{v, w\} \in F$ such that $\Upsilon(e) \in \{3,4\}$ we add edges $\{u_e, v_{\Upsilon(e)-2}\}$ and $\{u_e, w_{\Upsilon(e)-2}\}$ to E_3.



(a) A cubic graph H. The edge labels correspond to the 4-edge coloring.



(b) The underlying graph $G_{\downarrow}$. Big white vertices correspond to edges of H, black vertices correspond to vertices of H.



(c) The temporal graph $\mathcal{G}$.

Figure 5.3: Example of the reduction from INDEPENDENT SET on graphs with maximum degree three to TEMPORAL MATCHING.

The construction is illustrated in Figure 5.3. It is easy to check that vertices $u_e \in V$ with $e \in F$ have degree two in the underlying graph and vertices $v_1, v_2 \in V$ with $v \in U$ have degree at most three in the underlying graph, Furthermore, every edge in $\mathcal{G}$ appears only at one time step. Finally, we set $\Delta = 2$ and $k = h + |F|$.

Correctness. Next, we show that $\mathcal{G}$ contains a 2-temporal matching of size k if and only if H contains an independent set of size h.

($\Rightarrow$): Let M be a 2-temporal matching in $\mathcal{G}$ of size k. We first show that we can assume without loss of generality that M contains at most one of the time-edges $(\{v_1, v_2\}, 2)$ and $(\{w_1, w_2\}, 2)$ for every $\{v, w\} \in F$. This will allow us to construct an independent set for the original graph H from the temporal matching. Formally,

we show that if M is a 2-temporal matching of $\mathcal{G}$, then there exists a 2-temporal matching M' of $\mathcal{G}$ such that $|M'| = |M|$, and for every edge $e = \{v, w\} \in F$ the matching M' contains at most one of the time-edges $(\{v_1, v_2\}, 2)$ and $(\{w_1, w_2\}, 2)$.

Let $e = \{v, w\} \in F$ be an edge such that both $(\{v_1, v_2\}, 2)$ and $(\{w_1, w_2\}, 2)$ are in M. Without loss of generality we assume that $\Upsilon(e) = 1$. Since the lifetime of $\mathcal{G}$ is three and $(\{v_1, v_2\}, 2) \in M$, no time-edge in M different from $(\{v_1, v_2\}, 2)$ is incident with v_1 or v_2. Similarly, no time-edge in M different from $(\{w_1, w_2\}, 2)$ is incident with w_1 or w_2. In particular, we have that $(\{u_e, v_1\}, 1) \notin M$ and $(\{u_e, w_1\}, 1) \notin M$. Hence, the temporal matching M' which is obtained from M by replacing $(\{v_1, v_2\}, 2)$ with $(\{u_e, v_1\}, 1)$ is a 2-temporal matching of $\mathcal{G}$ with $|M'| = |M|$, and the number of edges $\{v, w\} \in F$ such that M' contains both $(\{v_1, v_2\}, 2)$ and $(\{w_1, w_2\}, 2)$ is reduced by one (in comparison to M). Repeating this process eventually leads to a 2-temporal matching M' with the desired property.

From now on we assume without loss of generality that if $\{v, w\} \in F$, then M contains at most one of the time-edges $(\{v_1, v_2\}, 2)$ and $(\{w_1, w_2\}, 2)$. We set $S = \{v \in U \mid (\{v_1, v_2\}, 2) \in M\}$ and we claim that S is an independent set of size at least h in H. The above assumption already implies that S is an independent set, hence we show in the following that is has the correct minimum size.

Notice that for every edge $e \in F$ the underlying graph $G_\downarrow$ of $\mathcal{G}$ contains exactly two edges incident with u_e and both of them appear in the same time step. Hence M can contain at most one time-edge incident with w_e, and therefore $|S| \geq |M| - |F| \geq k - |F| = h$.

($\Leftarrow$): Let $S \subseteq U$ with $|S| \geq h$ be an independent set of H. We show that $\mathcal{G}$ contains a 2-temporal matching M of size at least k. We construct M as follows. First, for every $v \in S$ we add $(\{v_1, v_2\}, 2)$ to M. Second, for every edge $e = \{v, w\} \in F$ we add one more time-edge in M as follows. Since S is independent, at least one of v and w is not in S, say v. Then we add to M

1. $(\{u_e, v_1\}, 1)$ if $\Upsilon(e) = 1$,
2. $(\{u_e, v_2\}, 1)$ if $\Upsilon(e) = 2$,
3. $(\{u_e, v_1\}, 3)$ if $\Upsilon(e) = 3$, and
4. $(\{u_e, v_2\}, 3)$ if $\Upsilon(e) = 4$.

By construction we have $|M| = |S| + |F|$. Now we show that M is a 2-temporal matching. For any two distinct vertices v and w in S the edges $\{v_1, v_2\}$ and $\{w_1, w_2\}$ do not share a vertex and therefore the time-edges $(\{v_1, v_2\}, 2)$ and $(\{w_1, w_2\}, 2)$ are not in conflict.

Furthermore, for any pair of adjacent edges $\{u_e, v_\alpha\}$, $\{v_1, v_2\}$ with $\alpha \in [2]$ in $E(G_\downarrow)$ the corresponding time-edges are not in conflict in M, as, by construction, at most one of them is contained in M. For the same reason, for every edge $e = \{v, w\} \in F$ the time-edges corresponding to $\{u_e, v_\alpha\}$ and $\{u_e, w_\alpha\}$, where $\alpha = 1 + (\Upsilon(e) + 1) \bmod 2$, are not in conflict in M. It remains to show that the time-edges $(\{w_e, u_\alpha\}, i)$ and $(\{w_{e'}, u_\alpha\}, j)$ with $i, j \in \{1,3\}$ and $\alpha \in [2]$ are not in conflict in M. Suppose to the contrary that the time-edges are in conflict. Then both of them are in M and $|i - j| \leq 1$. Since by definition $i, j \in \{1,3\}$, we conclude that $i = j$, that is, the time-edges appear in the same time step. Notice that e and e' share vertex u, and hence $\Upsilon(e) \neq \Upsilon(e')$. Since $\alpha = 1 + (\Upsilon(e) = 1) \bmod 2 = 1 + (\Upsilon(e') + 1) \bmod 2$, we conclude that either $\{\Upsilon(e), \Upsilon(e')\} = \{1,3\}$, or $\{\Upsilon(e), \Upsilon(e')\} = \{2,4\}$. However, by construction, this contradicts the assumption that $i = j$. This completes the proof that M is a 2-temporal matching and furthermore we have $|M| = |S| + |F| \geq k$. □

We observe that the reduction from the proof of Theorem 5.3 can be modified in such a way that it produces a temporal graph that has a complete underlying graph. Namely, we can add $\Delta = 2$ additional layers to the construction, one trivial layer at time step four, and one layer that is a complete graph at time step five. This has the consequence that the size of the matching increases by exactly $\lfloor |V|/2 \rfloor$ and the underlying graph of the constructed temporal graph is a complete graph. Hence, we obtain the following corollary.

Corollary 5.4. Temporal Matching *is NP-complete for all* $\Delta \geq 2$ *and* $\ell \geq 2\Delta + 1$ *even if the underlying graph of the input temporal graph is complete.*

This corollary implies that parameterizing Temporal Matching by structural graph parameters of the underlying graph that are constant on complete graphs cannot yield fixed-parameter tractability unless P = NP, even if combined with the lifetime ℓ.

We further remark that the reduction from the proof of Theorem 5.3 implies that the canonical maximization variant of Temporal Matching, where we want to maximize the cardinality of the temporal matching, is APX-hard [Mer+20].

We also remark that our reduction can easily be adapted to the model of Baste, Bui-Xuan, and Roux [BBR20]: recall that every edge of the underlying graph of the temporal graph constructed in the reduction (see proof of Theorem 5.3) appears in exactly one time step. Hence, for each of these time-edges, we can add a second appearance exactly one time step after the first appearance without creating any new matchable edges. Of course in order to do that for time-edges appearing in the third

time step, we need another fourth time step. It follows that γ-MATCHING [BBR20] is NP-hard and its canonical optimization version is APX-hard even if $\gamma = 2$ and $\ell = 4$.

Finally, we show how we can use the reduction from the proof of Theorem 5.3 to derive a kernelization lower bound for TEMPORAL MATCHING when parameterized by the number $|V|$ of vertices. In particular, this implies that the fixed-parameter tractability result for the parameter combination $\Delta + \mathrm{vc}_{\downarrow}$ [Mer+20, Theorem 34], where $\mathrm{vc}_{\downarrow}$ denotes the vertex cover number of the underlying graph, presumably cannot be improved to yield a polynomial kernel.

Proposition 5.5. TEMPORAL MATCHING *parameterized by the number $|V|$ of vertices does not admit a polynomial kernel for all $\Delta \geq 2$ unless NP $\subseteq$ coNP/poly.*

Proof. We provide an AND-cross-composition (for a definition see Section 2.3) from INDEPENDENT SET on graphs with maximum degree three [GJ79, GJS76]. Intuitively, we can just string together instances produced by the reduction we presented in the proof of Theorem 5.3 in the time axis such that the large instance contains a large Δ-temporal matching if and only if all original instances are YES-instances.

In this problem we are asked to decide whether a given graph $H = (U, F)$ with maximum degree three contains a set of at least h pairwise non-adjacent vertices. Furthermore, it is important to observe that, given graph $H = (U, F)$ with maximum degree three, it is NP-complete to decide whether H contains an independent set of size h even if it is known that H does not contain an independent set of size $h+1$ [GJS76]. In the following, we assume that all instances have this property. We define an equivalence relation R as follows: Two instances $(H = (U, F), h)$ and $(H' = (U', F'), h')$ are equivalent under R if and only if the number of vertices is the same, that is, $|U| = |U'|$ and we have that $h = h'$. Clearly, R is a polynomial equivalence relation.

Now let $(H_1 = (U_1, F_1), h_1), \ldots, (H_n = (U_n, F_n), h_n)$ be R-equivalent instances of INDEPENDENT SET with the above described extra conditions. We arbitrarily identify the vertices of all instances, that is, let $U = U_1 = \ldots = U_n$. For each (H_i, h_i) with $i \in [n]$ we construct an instance of TEMPORAL MATCHING as defined in the proof of Theorem 5.3 (for an illustration see Figure 5.3) with the only difference that we add a fourth layer that does not contain any edges. Now we put all constructed temporal graphs next to each other in temporal order, that is, if $\mathcal{G}^{(i)} = (V^{(i)}, E_1^{(i)}, E_2^{(i)}, \ldots, E_4^{(i)})$ is the graph constructed for (H_i, h_i), then the overall temporal graph is $\mathcal{G} = (\bigcup_{i \in [n]} V^{(i)}, E_1^{(1)}, E_2^{(1)}, \ldots, E_4^{(1)}, E_1^{(2)}, E_2^{(2)}, \ldots, E_4^{(2)}, \ldots, E_1^{(n)}, E_2^{(n)}, \ldots, E_4^{(n)})$. Note that $|\bigcup_{i \in [n]} V^{(i)}| \leq 2|U| + \binom{|U|}{2}$ since in the reduction defined in the proof of Theorem 5.3, the constructed temporal graph contains two vertices for every vertex of the

INDEPENDENT SET instance and one vertex for every edge of the INDEPENDENT SET instance. Further, we set $\Delta = 2$ and $k = n \cdot h_1 + \sum_{i \in [n]} |F_i|$.

This instance can be constructed in polynomial time and $|V|$ is polynomially upper-bounded by the maximum size of an input instance. It is easy to check that the extra trivial layer contained in each constructed temporal graph $\mathcal{G}^{(i)}$ prevents the Δ-temporal matchings from two adjacent constructed graphs $\mathcal{G}^{(i)}$ and $\mathcal{G}^{(i+1)}$ for $i \in [n-1]$ to interfere, that is, matching two vertices with a time edge from $\mathcal{G}^{(i)}$ cannot block vertices from $\mathcal{G}^{(i+1)}$ from being matched. Furthermore, since we assume that no instance (H_i, h_i) of INDEPENDENT SET contains an independent set of size $h_1 + 1$, it cannot happen that the Δ-temporal matching of a constructed temporal graph $\mathcal{G}^{(i)}$ is larger than $h_1 + |F_i|$. It follows from the proof of Theorem 5.3 that the constructed TEMPORAL MATCHING instance is a YES-instance if and only if for every $i \in [n]$ the INDEPENDENT SET instance (H_i, h_i) is a YES-instance.

Since INDEPENDENT SET is NP-hard under the above described restrictions [GJS76] and we AND-cross-composed it into TEMPORAL MATCHING with $\Delta = 2$ parameterized by $|V|$, this proves the proposition. □

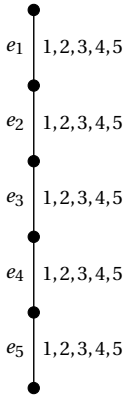
5.4 NP-Hardness of Temporal Matching with Underlying Paths

In this section we show NP-completeness of TEMPORAL MATCHING for $\Delta = 2$ even if the underlying graph of the input temporal graph is a path. In particular, this implies that in the FPT-algorithm for the parameter combination $\Delta + \mathrm{vc}_\downarrow$ [Mer+20, Theorem 34], where $\mathrm{vc}_\downarrow$ denotes the vertex cover number of the underlying graph, we presumably cannot replace the parameter $\mathrm{vc}_\downarrow$ with some smaller parameter such as the feedback vertex number of the underlying graph.

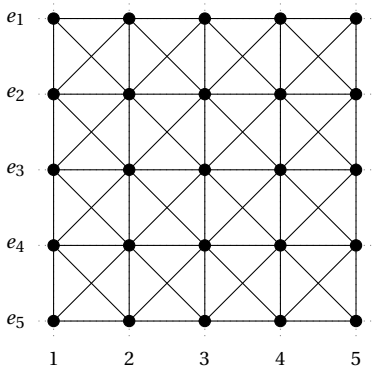
Theorem 5.6. TEMPORAL MATCHING *is NP-complete for all* $\Delta \geq 2$ *even if the underlying graph of the input temporal graph is a path.*

We show Theorem 5.6 by a polynomial-time reduction from INDEPENDENT SET on connected cubic planar graphs, which is known to be NP-complete [GJ77, GJ79]. More specifically, we show that INDEPENDENT SET is NP-complete on the temporal line graphs of temporal graphs that have a path as underlying graph. Recall that, by Observation 5.1, solving INDEPENDENT SET on a temporal line graph is equivalent to solving TEMPORAL MATCHING on the corresponding temporal graph. We proceed in the following steps.

1. We show that 2-temporal line graphs of temporal graphs that have a path as underlying graph have a grid-like structure. More specifically, we show



(a) Temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ with $G_\downarrow = P_6$, $\ell = 5$, and $E_1 = \ldots = E_5 = E(P_6)$. Edge names are on the left, time stamps of edges are listed on the right.



(b) 2-Temporal line graph $L_2(\mathcal{G})$. The horizontal dimension corresponds to time steps 1 to 5, the vertical dimension corresponds to the edges of the P_6.

Figure 5.4: A temporal line graph with a path as underlying graph where edges are always present and its 2-temporal line graph.

that they are induced subgraphs of so-called *diagonal grid graphs* or *king's graphs*[10] [Cha13, GZW18].

2. We show that INDEPENDENT SET is NP-complete on induced subgraphs of diagonal grid graphs which, together with Observation 5.1, yields Theorem 5.6.
   - We exploit that cubic planar graphs are induced topological minors of grid graphs and extend this result by showing that they are also induced topological minors of diagonal grid graphs.
   - We show how to modify the subdivision of a cubic planar graph that is an induced subgraph of a diagonal grid graph such that NP-hardness of finding independent sets of certain size is preserved.

We first give a formal definition of diagonal grid graphs or king's graphs. They are grid graphs that additionally have diagonal edges in every grid cell. Recall the definition of (normal) grid graphs.

[10] The name "king's graph" stems from the fact that the graph represents all legal moves of the king chess piece on a chessboard where each vertex represents a square on a chessboard and each edge is a legal move.

Definition 5.3 (Grid Graph)**.** A *grid graph* $Z_{n,m}$ has a vertex $v_{i,j}$ for all $i \in [n]$ and $j \in [m]$ and there is an edge $\{v_{i,j}, v_{i',j'}\}$ if and only if $|i-i'|+|j-j'| \leq 1$.

With a slight modification we arrive at the definition of diagonal grid graphs. An example for a diagonal grid graph is shown in Figure 5.4b.

Definition 5.4 (Diagonal Grid Graph [Cha13, GZW18])**.** A *diagonal grid graph* $\widehat{Z}_{n,m}$ has a vertex $v_{i,j}$ for all $i \in [n]$ and $j \in [m]$ and there is an edge $\{v_{i,j}, v_{i',j'}\}$ if and only if $|i-i'|^2+|j-j'|^2 \leq 2$.

We remark that diagonal grid graphs can also be characterized as the so-called strong product of two paths [BKZ05].

It is easy to check that for a temporal graph with a path as underlying graph and where each edge is active at every time step, the 2-temporal line graph is a diagonal grid graph. For a visualization see Figure 5.4.

Observation 5.7. *Let $P_n = (V,E)$ and $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$ with $E_i = E$ for all $i \in [\ell]$, then $L_2(\mathcal{G}) = \widehat{Z}_{n-1,\ell}$.*

Further, it is easy to see that removing an edge at a certain point in time results in removing the corresponding vertex from the diagonal grid graph. See Figure 5.5 for an example. Hence, we have that every induced subgraph of a diagonal grid graph is a 2-temporal line graph.

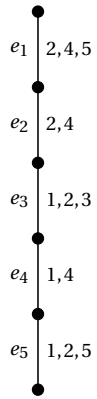
Corollary 5.8. *Let Z' be a connected induced subgraph of $\widehat{Z}_{n,m}$. Then there is an $n' \leq n$, an $\ell \leq m$, and a $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$ with $G_\downarrow = P_{n'}$ such that $Z' = L_2(\mathcal{G})$.*

Having these results at hand, it suffices to show that Independent Set is NP-complete on induced subgraphs of diagonal grid graphs. By Observation 5.1, this directly implies that Temporal Matching is NP-complete on temporal graphs that have a path as underlying graph. Hence, in the remainder of this section, we show the following result.

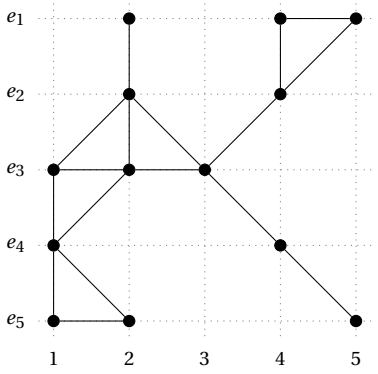
Theorem 5.9. Independent Set *on induced subgraphs of diagonal grid graphs is NP-complete.*

Theorem 5.9 may be of independent interest and strengthens a result of Clark, Colbourn, and Johnson [CCJ90] who showed that Independent Set is NP-complete on unit disk graphs. It is easy to see from Definition 5.4 that diagonal grid graphs and their induced subgraphs are a (proper) subclass of unit disk graphs.

The first building block for the reduction is the fact that we can embed cubic planar graphs into a grid [Val81]. More specifically, a cubic planar graph admits a planar



(a) Temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ with $G_\downarrow = P_6$, $\ell = 5$, and E_i for $i \in [5]$ as visualized. Edge names are on the left, time stamps of edges are listed on the right.



(b) 2-Temporal line graph $L_2(\mathcal{G})$. The horizontal dimension corresponds to time steps 1 to 5, the vertical dimension corresponds to the edges of the P_6.

Figure 5.5: A temporal line graph with a path as underlying graph where edges are *not* always active and its 2-temporal line graph.

embedding in such a way that the vertices are mapped to points of a grid and the edges are drawn along the grid lines. Moreover, such an embedding can be computed in polynomial time and the size of the grid is polynomially upper-bounded in the size of the planar graph.

Note that if we replace the edges of the original planar graph by paths of appropriate length, then the embedding in the grid is actually a subgraph of the grid. Furthermore, if we scale the embedding by a factor of two, that is, subdivide every edge once, then the embedding is also guaranteed to be an *induced* subgraph of the grid. In other words, we argue that every cubic planar graph is an induced topological minor of a polynomially large grid graph.

Proposition 5.10 (Special case of Theorem 2 from Valiant [Val81])**.** *Let $G = (V, E)$ be a cubic planar graph. Then G is an induced topological minor of $Z_{n,m}$ for some n, m with $n \cdot m \in O(|V|^2)$ and the corresponding subdivision of G can be computed in polynomial time.*

We discuss next how to replace the edges of a cubic planar graph by paths of appropriate lengths such that it is an induced subgraph of a *diagonal* grid graph. In

other words, we show that every cubic planar graph is an induced topological minor of a polynomially large diagonal grid graph.

Lemma 5.11. *Let $G = (V, E)$ be a cubic planar graph. Then G is an induced topological minor of $\widehat{Z}_{n,m}$ for some n, m with $n \cdot m \in O(|V|^2)$ and the corresponding subdivision of G can be computed in polynomial time.*

Proof. Let $G = (V, E)$ be a cubic planar graph. By Proposition 5.10 we know that there are integers n, m with $n \cdot m \in O(|V|^2)$ such that $G = (V, E)$ is an induced topological minor of $Z_{n,m}$. Let $G' = (V', E')$ with $V' \subseteq \mathbb{N} \times \mathbb{N}$ be the corresponding subdivision of G that is an induced subgraph of $Z_{n,m}$, that is, $Z_{n,m}[V'] = G'$. Furthermore, for each vertex $v \in V$ of G, let $v' \in V'$ denote the corresponding vertex in the subdivision G'.

Let $G'' = (V'', E'')$ be the graph resulting from subdividing each edge in G' eleven additional times and shift the graph three units away from the boundary of $Z_{n,m}$ in both dimensions. Intuitively, this is necessary to ensure that all paths in the grid are sufficiently far away from each other, which is also important in a later modification.

More formally, for each vertex $(i, j) \in V'$ create a vertex $(12i+3, 12j+3) \in V''$. For each edge $\{(i, j), (i, j+1)\} \in E'$ create eleven additional vertices, one for each grid point on the line between $(12i+3, 12j+3)$ and $(12i+3, 12j+15)$. We connect these vertices by edges such that we get an induced path on the new vertices together with vertices $(12i+3, 12j+3)$ and $(12i+3, 12j+15)$ that follows the grid line they lie on. For each edge $\{(i, j), (i+1, j)\} \in E'$ we make an analogous modification to G''. Furthermore, for each vertex $v \in V$ of G, let $v'' \in V''$ denote the corresponding vertex in the subdivision G''. It is clear that G'' is an induced subgraph of $Z_{12n+6,12m+6}$. We now show how to further modify G'' such that it is an induced subgraph of the diagonal grid graph $\widehat{Z}_{12n+6,12m+6}$.

For each vertex $v \in V$ let $v'' = (i, j) \in V''$. We check the following.

1. If $\deg_{G''}((i, j)) = 2$ and $\{(i, j), (i, j+1)\}, \{(i, j), (i+1, j)\}, \{(i+1, j), (i+2, j)\} \in E''$, then we delete $(i+1, j)$ from V'' and all its incident edges from E''. We add vertex $(i+1, j-1)$ to V'' and add edges $\{(i, j), (i+1, j-1)\}$ and $\{(i+1, j-1), (i+2, j)\}$ to E''. This modification is illustrated in Figure 5.6a. Rotated versions of this configuration are modified analogously.

2. If $\deg_{G''}((i, j)) = 3$ and $\{(i, j), (i, j+1)\}, \{(i, j), (i+1, j)\}, \{(i+1, j), (i+2, j)\}, \{(i, j), (i-1, j)\}, \{(i-1, j), (i-2, j)\} \in E''$, then we delete $(i+1, j)$ from V'' and all its incident edges from E''. We add vertex $(i+1, j-1)$ to V'' and add edges $\{(i, j), (i+1, j-1)\}$ and $\{(i+1, j-1), (i+2, j)\}$ to E''. Furthermore, we delete $(i-1, j)$ from V'' and all its incident edges from E''. We add vertex $(i-1, j-1)$ to V'' and add edges

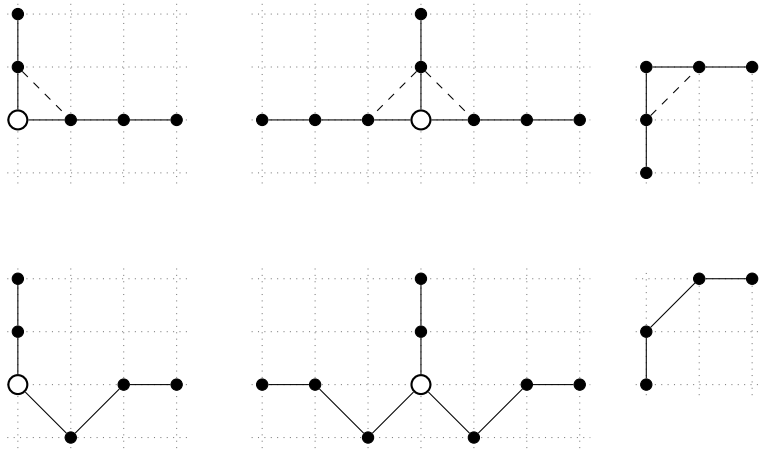


Figure 5.6: Illustration of the modifications described in the proof of Lemma 5.11. The situation before the modification is depicted above, dashed edges show unwanted edges present in an induced subgraph of a diagonal grid graph. The situation after the modification is depicted below.

$\{(i,j),(i-1,j-1)\}$ and $\{(i-1,j-1),(i-2,j)\}$ to E''. This modification is illustrated in Figure 5.6b. Rotated versions of this configuration are modified analogously.

Lastly, whenever a path in G'' that corresponds to an edge in G bends at a square angle, we remove the corner vertex and its incident edges and reconnect the path by a diagonal edge.

More formally, let $(i,j-1),(i,j),(i+1,j) \in V''$ be adjacent vertices in a path in G'' that corresponds to an edge in G, then we remove (i,j) from V'' and all its incident edges, and add the edge $\{(i,j-1),(i+1,j)\}$ to E''. This modification is illustrated in Figure 5.6c. Rotated versions of this configuration are modified analogously.

Now it is easy to see that G'' is an induced subgraph of $\widehat{Z}_{12n+6,12m+6}$. Furthermore, G'' can be computed in polynomial time. □

Next we argue that we can always embed a cubic planar graph into a diagonal grid graph in a way that preserves NP-hardness of Independent Set. This is based on the observation that subdividing an edge of a graph twice increases the size of a maximum independent set exactly by one.

Observation 5.12 (Poljak [Pol74])**.** *Let $G = (V, E)$ be a graph. Then for every $\{u, v\} \in E$, the graph $G' = (V \cup \{u', v'\}, (E \setminus \{\{u, v\}\}) \cup \{\{u, u'\}, \{u', v'\}, \{v', v\}\})$ contains an independent set of size $k+1$ if and only if G contains an independent set of size k.*

This observation implies that if we can guarantee that for every cubic planar graph there is a subdivision that subdivides every edge an even number of times and that is an induced subgraph of a diagonal grid graph of polynomial size, then we are done.

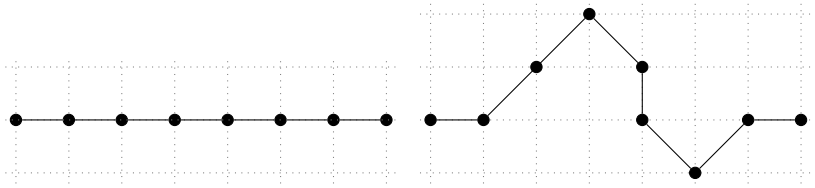
Lemma 5.13. *Let $G = (V, E)$ be a cubic planar graph. Then there is a subdivision of G that is an induced subgraph of $\widehat{Z}_{n,m}$ for some n, m with $n \cdot m \in O(|V|^2)$ and where each edge of G is subdivided an even number of times. Furthermore, the subdivision of G can be computed in polynomial time.*

Proof. Let $G = (V, E)$ be a cubic planar graph. By Lemma 5.11 we know that there are some n, m with $n \cdot m \in O(|V|^2)$ such that $G = (V, E)$ is an induced topological minor of $\widehat{Z}_{n,m}$. Let $G' = (V', E')$ with $V' \subseteq \mathbb{N} \times \mathbb{N}$ be a subdivision of G constructed in polynomial time as described in the proof of Lemma 5.11.

Recall that every edge e in G is replaced by a path P_e in G'. Observation 5.12 implies that if we can guarantee that all these paths have an odd number of edges (and hence result from an even number of subdivisions), then G' contains an independent set of size $k + \sum_{e \in E} \lfloor \frac{|E(P_e)|-1}{2} \rfloor$ if and only if G contains an independent set of size k. In the following we show how to change the parity of the number of edges of a path P_e in G' that corresponds to an edge e in G.

The number of subdivisions performed in the construction we described in the proof of Lemma 5.11 ensures that each path P_e in G' that corresponds to an edge e in G contains seven consecutive edges that are either all horizontal or all vertical. Assume that P_e contains an even number of edges and contains horizontal edges $\{(i, j), (i+1, j)\}, \{(i+1, j), (i+2, j)\}, \{(i+2, j), (i+3, j)\}, \{(i+3, j), (i+4, j)\}, \{(i+4, j), (i+5, j)\}, \{(i+5, j), (i+6, j)\}, \{(i+6, j), (i+7, j)\}$. We remove vertices $(i+2, j), (i+3, j), (i+5, j)$ and all their incident edges. We add vertices $(i+2, j+1), (i+3, j+2), (i+4, j+1), (i+5, j-1)$ and edges $\{(i+1, j), (i+2, j+1)\}, \{(i+2, j+1), (i+3, j+2)\}, \{(i+3, j+2), (i+4, j+1)\}, \{(i+4, j+1), (i+4, j)\}, \{(i+4, j), (i+5, j-1)\}, \{(i+5, j-1), (i+6, j)\}$. It is easy to check that this reconnects the path and increases the number of edges by one. This modification is illustrated in Figure 5.7. The vertical version of this configuration is modified analogously.

Using this modification we can easily modify G' in polynomial time in a way that all paths corresponding to edges of G have an odd number of edges. The result then follows. □



(a) Before. **(b)** After.

Figure 5.7: Illustration of the modification described in the proof of Lemma 5.13. It shows how to increase the length of an induced path of a diagonal grid graph by one.

This concludes the proof of Theorem 5.9. We can see that it follows directly from Lemma 5.13 and Observation 5.12. Finally, Theorem 5.9, Observation 5.1, and Corollary 5.8 together imply Theorem 5.6.

Theorem 5.6 implies that parameterizing Temporal Matching by structural graph parameters of the underlying graph that are constant on a path cannot yield fixed-parameter tractability unless P = NP, even if combined with Δ.

5.5 Conclusion

In this chapter, we provided an analysis of the computational complexity of Temporal Matching. As one of the main results, we showed that the problem remains NP-hard even if the underlying graph of the input temporal graph is a path. We proved this result by employing the concept of temporal line graphs, which is an interesting research topic by itself. Facing computational hardness even in quite restricted cases, the following issues remain research challenges.

In the domain of exact parameterized algorithms, in particular, treedepth of the underlying graph combined with Δ is left open. This would be a smaller parameter than vertex cover number of the underlying graph combined with Δ [Mer+20, Theorem 34] and it is unbounded in all known NP-hardness reductions.

Considering that Temporal Matching is APX-hard [Mer+20] and that we have NP-hardness even if the underlying graph is a path, the investigation of parameterized approximation algorithms seems promising. In fact, for the very restricted case that the underlying graph is a path, we can use known polynomial-time approximation schemes for Independent Set on unit disk graphs [Hun+98, Mat98] since we know that in this case the temporal line graph is a unit disk graph. This might become a good base case for an FPT-approximation scheme with some distance-to-triviality parameterization.

CHAPTER 6

Temporal Coloring

Graph coloring is one of the most famous computational problems with applications in a wide range of areas such as planning and scheduling, resource allocation, and pattern matching. So far coloring problems are mostly studied on static graphs, which often stand in contrast to practice where data is inherently dynamic and subject to discrete changes over time. In this chapter we present a natural temporal extension of the classic graph coloring problem. Given a temporal graph and two natural numbers k and Δ, we ask for a coloring sequence for each vertex such that

1. in every sliding time window of Δ consecutive time steps, in which an edge is active, this edge is properly colored (that is, its endpoints are assigned two different colors) at least once during that time window, and

2. the total number of different colors is at most k.

This sliding window temporal coloring problem abstractly captures many realistic graph coloring scenarios in which the underlying network changes over time, such as dynamically assigning communication channels to moving agents. We present a thorough investigation of the computational complexity of this temporal coloring problem. More specifically, we prove strong computational hardness results even for two colors, complemented by exact FPT-algorithms and one parameterized approximation algorithm. We show that some of our algorithms are asymptotically almost optimal under the Exponential Time Hypothesis (ETH).

This chapter is based on the paper "Sliding window temporal graph coloring" by Mertzios, Molter, and Zamaraev [MMZ19].

6.1 Introduction

In this chapter we introduce and rigorously study a new, yet natural temporal extension of the classic COLORING problem, called SLIDING WINDOW TEMPORAL COLORING. Recall that in the classic COLORING problem, we are asked to color the vertices of a given graph with at most k colors such that the endpoints of every edge are colored differently. In SLIDING WINDOW TEMPORAL COLORING the input consists of a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two natural numbers Δ and k. At every time

step t, every vertex has to be assigned one color (this color can be different every time), under the following constraint: Every edge e has to be properly colored at least once during *every time window* of Δ consecutive time steps, and this must happen at a time step t in this window when e is present. Now the question is whether there exists such a temporal coloring over the whole lifetime of the input temporal graph that uses at most k colors.

Our temporal extension of the COLORING problem is motivated by applications in mobile sensor networks and in planning. Consider the following scenario: every mobile agent broadcasts information over a specific communication channel while it listens on all *other* channels. Thus, whenever two mobile agents are sufficiently close, they can exchange information only if they broadcast on different channels. We assume that agents can switch channels at any time. To ensure a high degree of information exchange, it makes sense to find a schedule of assigning broadcasting channels to the agents over time which minimizes the number of necessary channels, while allowing each pair of agents to communicate at least once within every small time window in which they are close to each other.

6.1.1 Related Work

Temporal extensions of the classic graph coloring problem have also been previously studied by Yu et al. [Yu+13] (see also Ghosal and Ghosh [GG15]) in the context of channel assignment in mobile wireless networks. In this problem, every edge has to be properly colored in every layer of the input temporal graph $\mathcal{G}$, while the goal is to minimize some linear combination of the total number of colors used and the number of color re-assignments on the vertices [Yu+13]. In this temporal coloring approach, the notion of time is only captured by the fact that the number of re-assignments affects the value of the target objective function, while the fundamental solution concept remains the same as in static graph coloring, that is, *every* individual (static) layer has to be properly colored. Using this, Yu et al. [Yu+13] presented generic methods to adapt known algorithms and heuristics from static graph coloring to deal with their new objective function. Other temporal extensions of the classic vertex and edge coloring problems have been recently studied by Vizing [Viz15]. Vizing considered only temporal graphs of lifetime two, and in his problems every object to color (vertex or edge) has to be colored in exactly one of the layers of the input temporal graph in such a way that any two objects that are assigned the same color in the same layer are not adjacent in this layer. The goal of the problems is to minimize the total number of used colors.

In contrast to the model of Yu et al. [Yu+13], the solution concept in SLIDING WINDOW TEMPORAL COLORING is fundamentally different to that of static graph col-

oring as it takes into account the inherent dynamic nature of the temporal network. Indeed, even to verify whether a given solution is feasible, it is not sufficient to just consider every layer independently.

On static graphs, COLORING is a classic and very well-studied problem. For an overview on results for COLORING we refer to the monograph of Jensen and Toft [JT11].

6.1.2 Our Contributions and Organization of the Chapter

In this chapter we present a thorough investigation of the computational complexity of SLIDING WINDOW TEMPORAL COLORING. All notation specific to this chapter and the formal definition of the temporal problems that we study are presented in Section 6.2. First we investigate in Section 6.3 an interesting special case of SLIDING WINDOW TEMPORAL COLORING, called TEMPORAL COLORING, where the length Δ of the sliding time window is equal to the whole lifetime ℓ of the input temporal graph. We start by proving in Theorem 6.2 that TEMPORAL COLORING is NP-complete even for two colors, and even when every layer consists of one clique and isolated vertices. This is in stark contrast to the static coloring problem, where it can be decided in linear time whether a given (static) graph G is 2-colorable, that is, whether G is bipartite.

In Section 6.4 and in the reminder of the chapter we deal with the general version of SLIDING WINDOW TEMPORAL COLORING, where the value of Δ is arbitrary. On the one hand, we show that the problem is NP-hard even on very restricted special classes of input temporal graphs. On the other hand, we give an exponential-time algorithm for SLIDING WINDOW TEMPORAL COLORING that has an asymptotically optimal running time assuming the Exponential Time Hypothesis (ETH) whenever Δ is constant. Moreover we show how to extend the algorithm to get an FPT-algorithm for the parameter "number $|V|$ of vertices". Note that the assumption that $|V|$ is small while ℓ is large can be also reasonable in practical situations.

Finally, we consider in Section 6.4 an optimization variant of SLIDING WINDOW TEMPORAL COLORING where the number of colors is to be minimized. We give an FPT-approximation algorithm for the problem parameterized by the vertex cover number of the underlying graph $G_{\downarrow}$ that has an additive error of one (that is, uses at most one additional color). From a classification standpoint this is also optimal since the problem remains NP-hard to solve optimally on temporal graphs that have an underlying graph with a constant-size vertex cover.

6.1.3 Further Contributions of the Paper this Chapter is Based on

Additionally to the contributions we present in this chapter, Mertzios, Molter, and Zamaraev [MMZ19] show that TEMPORAL COLORING admits a polynomial kernel when parameterized by the number of vertices of the input temporal graph.

6.2 Preliminaries

In this section, we introduce further concepts related to temporal graphs and coloring and give the formal problem definitions of TEMPORAL COLORING and SLIDING WINDOW TEMPORAL COLORING.

6.2.1 Coloring

Given a static graph $G = (V, E)$, a *coloring* of G is a function $\Upsilon : V \to \mathbb{N}$ which assigns a color to every vertex in G. We say an edge $\{v, w\} \in E$ is *properly colored* if $\Upsilon(v) \neq \Upsilon(w)$. If $\Upsilon(v) = \Upsilon(w)$, then we say that the edge $\{v, w\}$ is colored *monochromatically*. A coloring Υ is *proper* if it colors every edge properly. The *size* of a coloring Υ is the number of colors it uses, that is, $|\Upsilon| := |\bigcup_{v \in V} \{\Upsilon(v)\}|$. We say that G is *k-colorable* if it admits a proper coloring Υ with $|\Upsilon| \leq k$.

6.2.2 Temporal Coloring

A *temporal coloring* of a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ is a function $\Upsilon : V \times [\ell] \to \mathbb{N}$, which assigns to every vertex appearance (v, t) with $t \in [\ell]$ in $\mathcal{G}$ one color $\Upsilon(v, t) \in \mathbb{N}$. The *size* of Υ is the total number $|\Upsilon| := |\bigcup_{v \in V, t \in [\ell]} \{\Upsilon(v, t)\}|$ of colors used by Υ. For every time step $t \in [\ell]$ we denote by Υ_t the restriction of Υ to the vertex appearances at time step t, that is, $\Upsilon_t : V \to \mathbb{N}$, such that $\Upsilon_t(v) = \Upsilon(v, t)$, for every $v \in V$. We refer to Υ_t as the *time step coloring* for the time step t. Furthermore, to ease the presentation, we will refer to the temporal coloring Υ as the ordered sequence $(\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_\ell)$ of all its time step colorings. Let $e \in E(G_\downarrow)$ be an edge of the underlying graph $G_\downarrow$. We say that an edge $e = \{u, v\}$ of the underlying graph $G_\downarrow$ is *properly temporally colored* at time step t if $\Upsilon_t(u) \neq \Upsilon_t(v)$ and $e \in E_t$, that is, the edge e is present at time step t. We are now ready to introduce the definition of a *proper temporal coloring*.

Definition 6.1 (Proper Temporal Coloring)**.** Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a temporal graph. A *proper temporal coloring* of $\mathcal{G}$ is a temporal coloring $\Upsilon = (\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_\ell)$ such that every edge $e \in E(G_\downarrow)$ is properly temporally colored in at least one time step $t \in [\ell]$.

Using this definition, we can formally define the decision problem TEMPORAL COLORING.

TEMPORAL COLORING	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and an integer $k \in \mathbb{N}$.
Question:	Is there a proper temporal coloring Υ of $\mathcal{G}$ using $\lvert\Upsilon\rvert \leq k$ colors?

Note that TEMPORAL COLORING is a natural extension of the classic NP-complete COLORING problem [GJ79, GJS76, Kar72] on static graphs to temporal graphs. In particular, COLORING is the special case of TEMPORAL COLORING where the lifetime of the input temporal graph is $\ell = 1$. Moreover, it is easy to see that it can be verified in polynomial time whether a given temporal coloring Υ is proper. Hence, we have that TEMPORAL COLORING is NP-complete for each fixed $k \geq 3$ and $\ell \geq 1$ (since COLORING is NP-hard for all $k \geq 3$ [GJ79, GJS76] and to get $\ell > 1$ we can add trivial layers to the temporal graph).

We remark that TEMPORAL COLORING can be treated as a *multi-layer graph* problem: It is easy to check that, given a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and an integer $k \in \mathbb{N}$, it holds that for every permutation $\pi : [\ell] \to [\ell]$ we have that $(\mathcal{G}, k)$ is a YES-instance of TEMPORAL COLORING if and only if $(\mathcal{G}' = (V, (E_{\pi(i)})_{i\in[\ell]}), k)$ is a YES-instance of TEMPORAL COLORING.

6.2.3 Sliding Δ-Window Temporal Coloring

In the definition of a proper temporal coloring given in Definition 6.1, we require that every edge is properly temporally colored at least once during the whole lifetime ℓ of the temporal graph $\mathcal{G}$. However, in many real-world applications, where ℓ is expected to be arbitrarily large, we may need to require that every edge is properly temporally colored more often, and in particular, at least once during every Δ-time window, for some given Δ, regardless of how large the lifetime ℓ is. We formalize this in the definition of a *proper sliding Δ-window temporal coloring*.

Definition 6.2 (Proper Sliding Δ-Window Temporal Coloring)**.** Let $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ be a temporal graph and let $\Delta \leq \ell$. A *proper sliding Δ-window temporal coloring* of $\mathcal{G}$ is a temporal coloring $\Upsilon = (\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_\ell)$ such that, for every Δ-window W_t^Δ and for every edge $e \in E_{W_t^\Delta}$ we have that e is properly temporally colored in at least one time step $t' \in W_t^\Delta$.

An example of a proper sliding Δ-window temporal coloring is given in Figure 6.1. Using this definition, we can formally define the decision problem SLIDING WINDOW TEMPORAL COLORING.

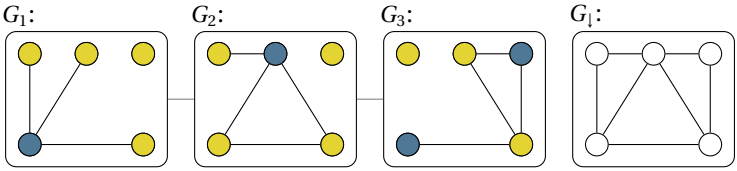


Figure 6.1: Example temporal graph with lifetime three and a proper sliding Δ-window temporal 2-coloring for $\Delta = 2$. Notice that for example one edge of G_2 is colored monochromatically, but this edge is also active at time steps one and three and is colored properly in the corresponding layers.

SLIDING WINDOW TEMPORAL COLORING	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.
Question:	Is there a proper sliding Δ-window temporal coloring Υ of $\mathcal{G}$ using $\lvert\Upsilon\rvert \leq k$ colors?

Note that the problem TEMPORAL COLORING defined above in this section is the special case of SLIDING WINDOW TEMPORAL COLORING where $\Delta = \ell$, that is, where there is only one Δ-window in the whole temporal graph. Moreover, it is easy to see that it can be verified in polynomial time whether a given temporal coloring Υ is a proper sliding Δ-window temporal coloring. Hence, we have that SLIDING WINDOW TEMPORAL COLORING is NP-complete for each fixed $k \geq 3$, $\Delta \geq 1$, and $\ell \geq \Delta$.

6.2.4 Basic Observations

We start with the observation that computational hardness of SLIDING WINDOW TEMPORAL COLORING for some fixed value of Δ implies hardness for all larger values of Δ. This allows us to construct hardness reductions for small fixed values of Δ and still obtain general hardness results.

Observation 6.1. *Let Δ be a fixed constant.* SLIDING WINDOW TEMPORAL COLORING *on instances $(\mathcal{G}, k, \Delta + 1)$ is computationally at least as hard as* SLIDING WINDOW TEMPORAL COLORING *on instances $(\mathcal{G}, k, \Delta)$.*

Proof. To see the correctness of Observation 6.1, we show that we can easily reduce from SLIDING WINDOW TEMPORAL COLORING with input Δ to SLIDING WINDOW TEMPORAL COLORING with input $(\Delta + 1)$ by inserting a trivial layer after every Δ consecutive layers. Let $(\mathcal{G}, k, \Delta)$ denote the original instance and $(\mathcal{G}', k, \Delta + 1)$ the constructed instance. For a visualization see Figure 6.2.

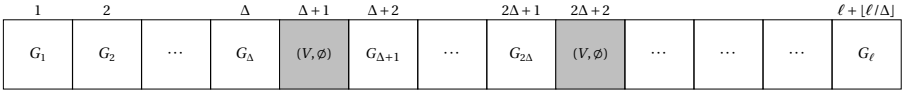


Figure 6.2: Inserting trivial layers to reduce SLIDING WINDOW TEMPORAL COLORING on instances $(\mathcal{G}, \Delta, k)$ to SLIDING WINDOW TEMPORAL COLORING on instances $(\mathcal{G}, \Delta+1, k)$.

($\Rightarrow$): If $\mathcal{G}$ admits a proper sliding Δ-window temporal coloring, then we can easily modify this coloring for $\mathcal{G}'$. The inserted trivial layers can be colored arbitrarily and all other layers are colored in the same way the corresponding layers from $\mathcal{G}$ are colored. This yields a proper sliding $(\Delta+1)$-window temporal coloring for $\mathcal{G}'$.

($\Leftarrow$): If $\mathcal{G}'$ admits a proper sliding $(\Delta+1)$-window temporal coloring, then we can easily modify this coloring for $\mathcal{G}$. We ignore how the inserted trivial layers are colored and color all layers of $\mathcal{G}$ in the same way the corresponding layers from $\mathcal{G}'$ are colored. This yields a proper sliding Δ-window temporal coloring for $\mathcal{G}$. □

6.3 Hardness Results for Temporal Coloring

In this section we investigate the parameterized computational complexity of TEMPORAL COLORING. We give two hardness results that in particular show that TEMPORAL COLORING is already NP-complete for two colors, even if the input temporal graph is very restricted. This stands in stark contrast to the static case, where checking whether a graph is 2-colorable can be done in linear time[11].

We start by showing that TEMPORAL COLORING is NP-complete even if each layer is a clique together with some isolated vertices. From a motivation standpoint, this excludes an interesting special case of the mobile agent scenario, where at each time step exactly one group of agents meet such that they can all pairwise communicate.

Theorem 6.2. TEMPORAL COLORING *is NP-complete for each fixed $k \geq 2$ even if each layer consists of one clique together with isolated vertices.*

We show this result for $k = 2$ and later give some arguments why our proof is easily adaptable for larger values of k. First we show the following lemma, which we will make use of in the proof of Theorem 6.2.

Lemma 6.3. *A graph G has two bipartite subgraphs that cover all edges of G if and only if G is 4-colorable.*

[11] This is a folklore result. One possible algorithm roughly works as follows. Colors are assigned in a preorder traversal of a depth-first-search forest, such that all forest edges are properly colored. If afterwards the graph contains a monochromatic edge, then it is not 2-colorable.

Proof. Assume that a given graph $G = (V, E)$ has two bipartite subgraphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ that cover all edges E of G, that is, $E = E_1 \cup E_2$. Let $\Upsilon_i : V \to \{1,2\}$ be the coloring of G_i for $i \in \{1,2\}$. Then $\Upsilon(v) := \pi(\Upsilon_1(v), \Upsilon_2(v))$ is a 4-coloring for G, where π is an arbitrary pairing function[12]: First note that $|\bigcup_{v \in V} \{\Upsilon(v)\}| \leq 4$ since $\Upsilon(v) \in \{\pi(1,1), \pi(1,2), \pi(2,1), \pi(2,2)\}$ for all $v \in V$. Now let $\{v, w\} \in E$. By assumption we have that $\{v, w\} \in E_1 \cup E_2$. Assume that $\{v, w\} \in E_1$ (the other case is analogous), then we have that $\Upsilon_1(v) \neq \Upsilon_1(w)$. It follows that $\Upsilon(v) \neq \Upsilon(w)$.

It remains to show that if a given graph $G = (V, E)$ is 4-colorable, then it has two bipartite subgraphs that cover all edges of G. Let $\Upsilon : V \to \{1,2,3,4\}$ be a 4-coloring for G. Let $E_1 := \{\{v, w\} \in E \mid \Upsilon(v) \in \{1,2\} \wedge \Upsilon(w) \in \{3,4\}\}$ and $E_2 := \{\{v, w\} \in E \mid \Upsilon(v) \in \{1,3\} \wedge \Upsilon(w) \in \{2,4\}\}$. It is easy to check that $G_1 = (V, E_1)$ is bipartite: One part is formed by vertices colored in 1 or 2 and the second part by vertices colored in 3 or 4. By definition E_1 does not contain edges between vertices from the same part. The argument for $G_2 = (V, E_2)$ is analogous. They are both subgraphs since $E_i \subseteq E$ for $i \in \{1,2\}$. It remains to show that $E = E_1 \cup E_2$: Let $\{v, w\} \in E$, then if $\{\Upsilon(v), \Upsilon(w)\} \in \{\{1,3\}, \{2,3\}, \{1,4\}, \{2,4\}\}$, then $\{v, w\} \in E_1$, otherwise (if $\{\Upsilon(v), \Upsilon(w)\} \in \{\{1,2\}, \{3,4\}\}$) we have that $\{v, w\} \in E_2$. □

Now we prove Theorem 6.2 for the case that $k = 2$.

Proof of Theorem 6.2 for $k = 2$. We give a polynomial-time reduction from the NP-complete 4-COLORING problem [GJ79, GJS76] where, given a graph H, we are asked to properly color H with four colors. Let $H = (U, F)$ be an instance of 4-COLORING. We construct a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ with $V = U$, $E_1 = E_2 = E(K_{|U|})$, $\ell = \binom{|U|}{2} - |F| + 2$, and for every non-edge of H there is exactly one time step i with $3 \leq i \leq \ell$ where only this edge is present. Note that every layer is either complete or only contains a single edge. Hence, every layer consists of a clique together with some isolated vertices.

Correctness. We now prove the correctness of our reduction, namely, show that the constructed temporal graph can be properly temporally colored with two colors if and only if the input graph is 4-colorable.

($\Rightarrow$): If H is 4-colorable, then we can use the 4-coloring of H to 2-color G_1 and G_2 using Lemma 6.3 and for every edge that is not present in G, color it properly in the layer where it is present.

($\Leftarrow$): If $\mathcal{G}$ is properly colorable with $k = 2$ colors, then all edges present in H have to be properly colored either in G_1 or G_2, that is, the edges of H can be covered by

[12] A function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a *pairing function* if it is a bijection.

two bipartite graphs, and hence, by Lemma 6.3, we can properly 4-color H. □

To adapt this proof for larger values of k, it is necessary to generalize Lemma 6.3 to the statement "A graph G has two k-colorable subgraphs that cover all edges of G if and only if G is k^2-colorable". It is easy to check that this can be done in an analogous way for each fixed k. Using this more general lemma, one can easily adapt the reduction in the proof of Theorem 6.2. We remark that, from a parameterized point of view, this result implies that parameterizing Temporal Coloring by structural graph parameters of the layers that are constant on a graph consisting of a clique with some isolated vertices cannot yield fixed-parameter tractability unless P = NP, even if combined with k.

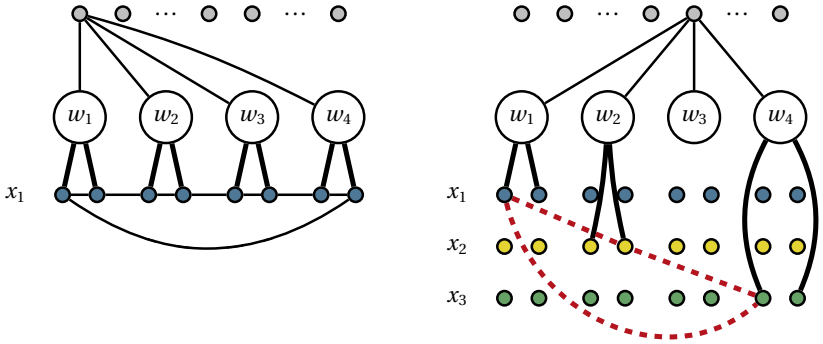
Now we show with a different reduction that Temporal Coloring remains hard even if each layer has very few edges and the underlying graph has small degeneracy.

Theorem 6.4. Temporal Coloring *is NP-complete for all $k \geq 2$ even if the number of edges in each layer is in $O(k^2)$, the degeneracy of the underlying graph is in $O(k)$ and the underlying graph has domination number four.*

Proof. We present a polynomial-time reduction from Exact (3,4)-SAT [Tov84] to Temporal Coloring with $k = 2$. The reduction can be easily modified to a larger number of colors, we explain how to do this at the end of the proof. Recall that in Exact (3,4)-SAT we are asked to decide whether a given Boolean formula ϕ is satisfiable and ϕ is in conjunctive normal form where every clause has exactly three distinct literals and every variable appears in exactly four clauses. Given a formula ϕ with n variables and m clauses, we construct a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ consisting of $\ell = (n + 2m)$ layers, that is, one layer for each variable gadget and two layers for each clause gadget. An illustration of the construction is given in Figure 6.3. We start by adding four vertices w_1, w_2, w_3, and w_4 which will help to encode the first, second, third, and fourth appearance of a variable.

Variable gadget. For each variable x_i with $i \in [n]$ of ϕ we create nine vertices $v_{x_i}^{(1)}, v_{x_i}^{(2)}, \ldots, v_{x_i}^{(8)}$ (which we also refer to as "the vertices corresponding to x_i"), and u_{x_i}, and one new layer. In this new layer, we connect $v_{x_i}^{(j)}$ with $v_{x_i}^{((j \bmod 8)+1)}$ for all $j \in [8]$ and we connect $v_{x_i}^{(2h-1)}$ and $v_{x_i}^{(2h)}$ with w_h for all $h \in [4]$. Furthermore, we connect u_{x_i} with w_1, w_2, w_3, and w_4. It is easy to check that every layer corresponding to a variable contains twenty edges. For a visualization of the variable gadget see Figure 6.3a.

Clause gadget. For each clause c_i with $1 \leq i \leq m$ of ϕ we add two new layers and one new vertex u_{c_i}. In the first new layer we connect it with w_1, w_2, w_3, and w_4.



(a) Variable gadget.

(b) First layer of clause gadget.

Figure 6.3: Illustration of the reduction from EXACT (3,4)-SAT to TEMPORAL COLORING of the proof of Theorem 6.4. Figure 6.3a depicts the variable gadget for x_1. Figure 6.3b depicts the first layer of the clause gadget for clause $(x_1 \vee \neg x_2 \vee x_3)$, where we have the first appearance of x_1 (blue), the second appearance of x_2 (yellow), and the fourth appearance of x_3 (green). The second layer of the clause gadget contains only the red dashed triangle and is not depicted. In both figures vertices corresponding to the remaining variables are not depicted. Thick edges are present in exactly two layers and thin edges are present in exactly one layer.

Let x_j be a variable that appears in clause c_i and let this be the hth appearance of x_j in ϕ. Then we connect w_h with $v_{x_j}^{(2h-1)}$ and $v_{x_j}^{(2h)}$ in the first new layer. Lastly, denote x_{j_1}, x_{j_2}, and x_{j_3} the three variables in c_i appearing for the h_1th, h_2th, and h_3th time, respectively, and let $y_s = 1$ if x_{j_s} appears non-negated in c_i and $y_s = 0$, otherwise. We pairwise connect $v_{x_{j_1}}^{(2h_1-y_1)}$, $v_{x_{j_2}}^{(2h_2-y_2)}$, and $v_{x_{j_3}}^{(2h_3-y_3)}$ in both the first and the second new layer, we refer to these three vertices as "the triangle corresponding to clause c_i". It is easy to check that every layer corresponding to a clause contains at most thirteen edges. For a visualization of the first layer of the clause gadget see Figure 6.3b.

Before we show correctness, let us check that the underlying graph of $\mathcal{G}$ has constant degeneracy. We can show this by using the following degeneracy ordering: First we order all vertices u_{x_i} and u_{c_i} arbitrarily and put them at the beginning of the degeneracy ordering. Then we order the vertices $v_{x_i}^{(j)}$ arbitrarily and put them next in the ordering. Lastly, we add vertices w_1, w_2, w_3, and w_4 to the ordering. Note that all vertices u_{x_i} and u_{c_i} for some variable x_i or clause c_i, respectively, have degree four since they are only connected to w_1, w_2, w_3, and w_4. The vertices $v_{x_i}^{(j)}$ have degree five: In the layer of the variable gadget for x_i they are connected to two other vertices

$v_{x_i}^{(j')}$ and $v_{x_i}^{(j'')}$ and to one of the vertices w_1, w_2, w_3, and w_4. Then, depending on the value of j, there is at most one "clause triangle" that contains $v_{x_i}^{(j)}$. Once all these vertices are removed from the graph, the vertices w_1, w_2, w_3, and w_4 are isolated. It follows that the degeneracy of $G_\mathbb{1}$ is at most five.

Furthermore, it is straightforward to check that the vertices w_1, w_2, w_3, and w_4 form a dominating set in the underlying graph.

Correctness. It is easy to check that the reduction can be computed in polynomial time. It remains to show that $\mathscr{G}$ admits a proper temporal 2-coloring if and only if ϕ is satisfiable.

($\Rightarrow$): Assume that we are given a satisfying assignment for ϕ. Then we construct a proper temporal 2-coloring for $\mathscr{G}$ as follows. Let the two colors be red and blue. Whenever we do not specify the color of vertices in a certain layer, those vertices can be colored arbitrarily in that layer. In each layer, we color all vertices u_{x_i} and u_{c_j} with $i \in [n]$ and $j \in [m]$ red, and vertices w_1, w_2, w_3, w_4 blue.

Now consider the layers corresponding to variable gadgets. If variable x_i is set to true in the satisfying assignment for ϕ, then we color (in the layer corresponding to the variable gadget for x_i) vertices $v_{x_i}^{(2h-1)}$ red and vertices $v_{x_i}^{(2h)}$ blue for $h \in [4]$. Otherwise we color the vertices exactly in the opposite way. This leaves exactly four edges monochromatic in each layer corresponding to a variable gadget. These will be colored properly in the four clause gadgets corresponding to the four clauses where the corresponding variable appears.

Next, consider the layers corresponding to clause gadgets, in particular the first layer corresponding to clause c_i. Let x_1, x_2, and x_3 be the three variables appearing in c_i and, without loss of generality, let x_1 be contained in a literal that satisfies the clause and let that be the hth appearance of x_1. If x_1 appears non-negated, then we color $v_{x_1}^{(2h-1)}$ blue and all other vertices corresponding to variables x_1, x_2, and x_3 red. Otherwise, we color $v_{x_1}^{(2h)}$ blue and all other vertices corresponding to variables x_1, x_2, and x_3 red. Since the literal containing x_1 satisfies clause c_i, we have that the edge between w_h and $v_{x_1}^{(2h-1)}$ or $v_{x_1}^{(2h)}$, respectively, is colored properly in the layer corresponding to the variable gadget of x_1. Hence all edges between w_1, w_2, w_3, w_4 and vertices corresponding to variables x_1, x_2, and x_3 are colored properly. Out of the edges that form the triangle corresponding to c_i in the layer corresponding to clause c_i, exactly one is colored monochromatically. We color the vertices of the triangle in the *second* layer corresponding to the variable clause of c_i such that exactly that edge is colored properly. It is easy to verify that this describes a proper temporal 2-coloring for $\mathscr{G}$.

($\Leftarrow$): Assume that we are given a proper temporal 2-coloring for $\mathscr{G}$. Then we

construct a satisfying assignment for ϕ in the following way: We start with the observation that in any proper temporal coloring, vertices w_1, w_2, w_3, and w_4 have the same color in each layer that corresponds to a variable gadget and in each first layer corresponding to a clause gadget. Further, in each layer corresponding to a variable gadget there is a cycle of size eight containing all vertices corresponding to the variable of that gadget. Let that variable be x_i. Since all edges involved in this cycle only exist in this one layer, there are exactly two ways to color this cycle. One of them leaves the edges between $v_{x_i}^{(2h-1)}$ and w_h monochromatic for $h \in [4]$. The other way to color the cycle is the inverse coloring and leaves the edges between $v_{x_i}^{(2h)}$ and w_h monochromatic for $h \in [4]$. In the first case, we set x_i to false, and in the second case we set x_i to true. We claim that this yields a satisfying assignment for ϕ.

Assume for contradiction that it does not. Then there is a clause that is not satisfied. Let that clause be c_i. Recall that in a proper coloring, also vertices w_1, w_2, w_3, and w_4 have the same colors in each first layer that corresponds to a clause gadget. Consider the triangle corresponding to clause c_i in the first layer of the clause gadget of c_i. We have that in a proper temporal coloring, this triangle cannot be monochromatic, since, otherwise, one of the three edges is not properly colored in any of the layers of the temporal graph. Note that the triangle edges only exist in the two layers corresponding to the clause gadget of c_i and in the second layer, not all three edges can be colored properly. Hence, in the first layer of the clause gadget, at least one of the vertices of the triangle corresponding to c_i has a different color than vertices w_1, w_2, w_3, and w_4. However, this means that the corresponding variable is set to a truth value that satisfies this clause—a contradiction.

Modification for a Larger Number of Colors. To modify this reduction for more colors we introduce new vertices and edges to all layers to "block" all colors except two from being used. Formally, we do the following. Let $k > 2$. For each layer $i \in [\ell]$, we add $k-2$ fresh vertices $c_1^{(i)}, \ldots, c_{k-2}^{(i)}$, connect them to form a clique in layer i, and connect them to all non-isolated vertices in layer i. In all layers different from i the vertices $c_1^{(i)}, \ldots, c_{k-2}^{(i)}$ are isolated. All new edges exist in exactly one layer and hence have to be colored properly in this layer. It follows that the vertices $c_1^{(i)}, \ldots, c_{k-2}^{(i)}$ have to be colored with $k-2$ distinct colors and these colors then cannot be used to color any other non-isolated vertex in layer i.

The modification introduces $\ell \cdot (k-2)$ new vertices to the temporal graph and it is easy to check that it introduces $O(k^2)$ new edges to each layer. The degeneracy of the underlying graph is in $O(k)$ since we can put all new vertices to the beginning of the degeneracy ordering described earlier in this proof, and it is easy to check that all new vertices have a degree in $O(k)$ in the underlying graph. Vertices w_1, w_2, w_3,

and w_4 still form a dominating set in the underlying graph since they are connected to all new vertices. □

We remark that Theorem 6.4 has some interesting implications from a parameterized point of view. Parameterizing TEMPORAL COLORING by structural graph parameters of the layers that are constant on graphs with constantly many edges cannot yield fixed-parameter tractability unless P = NP, even if combined with k.

6.4 Complexity of Sliding Window Temporal Coloring

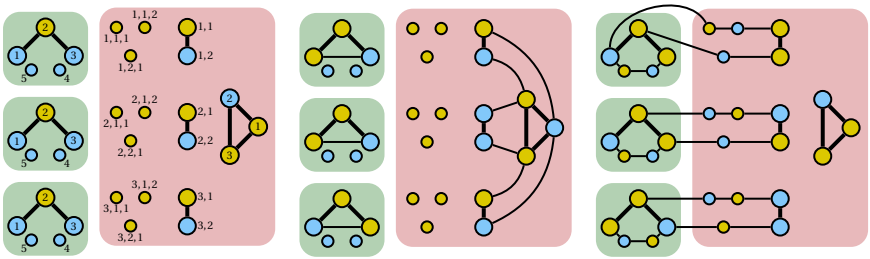
In this section we investigate the parameterized computational complexity of SLIDING WINDOW TEMPORAL COLORING. We first give a refined NP-hardness reduction together with an ETH lower bound. We give an exponential time algorithm that matches the lower bound for constant Δ and show how to extend this algorithm to obtain fixed-parameter tractability for SLIDING WINDOW TEMPORAL COLORING when parameterized by the number of vertices of the input temporal graph. In contrast to TEMPORAL COLORING we can show that SLIDING WINDOW TEMPORAL COLORING does not admit a polynomial kernel when parameterized by the number of vertices of the input temporal graph unless NP $\subseteq$ coNP/poly. We proceed by showing that SLIDING WINDOW TEMPORAL COLORING is NP-complete even if $k = 2$ and the underlying graph of the input temporal graph has a vertex cover number that only depends on k. Lastly, we show how to adapt our algorithm for SLIDING WINDOW TEMPORAL COLORING for a canonical optimization variant of the problem, where we want to minimize the number of colors. We achieve an FPT-approximation algorithm that uses at most one extra color for the parameter "vertex cover number of the underlying graph".

6.4.1 NP-Hardness Results

We now present the main computational hardness result of this section. In particular, we show that SLIDING WINDOW TEMPORAL COLORING is NP-complete for $k = 2$ even if the temporal input graph has only three layers.

Theorem 6.5. SLIDING WINDOW TEMPORAL COLORING *is NP-complete for all* $k \geq 2$, $\Delta \geq 2$, *and* $\ell \geq \Delta + 1$, *even if*

- *the underlying graph is* $(k+1)$*-colorable,*
- *the underlying graph has a maximum degree in* $O(k)$, *and*
- *every layer has connected components with size in* $O(k)$.



(a) Layer one. **(b)** Layer two. **(c)** Layer three.

Figure 6.4: Illustration of the reduction from EXACT (3,4)-SAT to SLIDING WINDOW TEMPORAL COLORING of the proof of Theorem 6.5. Vertices and edges in the red shaded areas (right) correspond to a clause gadget for clause $(\neg x_1 \vee x_2 \vee x_3)$. Vertices and edges in the green shaded areas (left) correspond to the variable gadgets for x_1, x_2, and x_3. Thick edges appear in every layer while thin edges only appear in one layer. The vertices are colored according to a coloring that would be constructed for the assignment $x_1 = \texttt{true}, x_2 = \texttt{true}, x_3 = \texttt{false}$. In the first layer (a), the superscripts of the vertices used in the proof of Theorem 6.5 are shown. To keep the figure clean, the superscripts are omitted in the illustrations for layers (b) and (c).

Proof. We present a polynomial-time reduction from EXACT (3,4)-SAT [Tov84] to SLIDING WINDOW TEMPORAL COLORING with $k = 2$ and $\Delta = 2$. The reduction can be easily modified to a larger number of colors, we explain how to do this at the end of the proof. Recall that in EXACT (3,4)-SAT we are asked to decide whether a given Boolean formula ϕ is satisfiable and ϕ is in conjunctive normal form where every clause has exactly three distinct literals and every variable appears in exactly four clauses. On an intuitive level, the main idea that we exploit in this reduction is that no matter how a triangle is colored with two colors, always (exaclty) one of the three edges is monochromatic. We use this idea both to construct variable gadgets (by further enforcing that a specific edge of the triangle always has to be properly colored) and to construct clause gadgets, where the three edges of a triangle correspond to the three literals in a clause and the monochromatic edge "selects" which literal should satisfy the clause.

Given a formula ϕ with n variables and m clauses, we construct a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ consisting of $\ell = 3$ layers, which we will refer to as $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $G_3 = (V, E_3)$. To increase the number of layers, we can repeat G_3. We construct the following variable gadgets and clause gadgets. An illustration of the construction is given in Figure 6.4.

Variable gadget. For each variable x_i with $i \in [n]$ of ϕ we create five vertices $v_{x_i}^{(1)}$, $v_{x_i}^{(2)}$, $v_{x_i}^{(3)}$, $v_{x_i}^{(4)}$, and $v_{x_i}^{(5)}$. The vertices $v_{x_i}^{(1)}$, $v_{x_i}^{(2)}$, and $v_{x_i}^{(3)}$ form a (not necessarily induced) P_3 in every layer, that is, $\{v_{x_i}^{(1)}, v_{x_i}^{(2)}\} \in E_t$ and $\{v_{x_i}^{(2)}, v_{x_i}^{(3)}\} \in E_t$ for all $t \in [3]$. Furthermore, we connect $v_{x_i}^{(1)}$ and $v_{x_i}^{(3)}$ in the second layer, that is, $\{v_{x_i}^{(1)}, v_{x_i}^{(3)}\} \in E_2$. Lastly, we create a full C_5 in layer three, that is, $\{v_{x_i}^{(3)}, v_{x_i}^{(4)}\} \in E_3$, $\{v_{x_i}^{(4)}, v_{x_i}^{(5)}\} \in E_3$, and $\{v_{x_i}^{(1)}, v_{x_i}^{(5)}\} \in E_3$.

Clause gadget. For each clause c_i with $i \in [m]$ of ϕ we create a total of 18 vertices. We create vertices $v_{c_i}^{(1)}$, $v_{c_i}^{(2)}$, and $v_{c_i}^{(3)}$ and connect them to form a triangle in every layer, that is, $\{v_{c_i}^{(1)}, v_{c_i}^{(2)}\} \in E_t$, $\{v_{c_i}^{(2)}, v_{c_i}^{(3)}\} \in E_t$, and $\{v_{c_i}^{(1)}, v_{c_i}^{(3)}\} \in E_t$ for all $t \in [3]$. In this proof, we refer to these vertices as the *core* of the clause gadget of clause c_i. Next, we add six vertices, which we refer to as the *extension* of the core of the clause gadget of clause c_i. Let these vertices be called $v_{c_i}^{(1,1)}$, $v_{c_i}^{(1,2)}$, $v_{c_i}^{(2,1)}$, $v_{c_i}^{(2,2)}$, $v_{c_i}^{(3,1)}$, and $v_{c_i}^{(3,2)}$. We connect $v_{c_i}^{(j,1)}$ and $v_{c_i}^{(j,2)}$ for all $j \in [3]$ in every layer, that is, $\{v_{c_i}^{(j,1)}, v_{c_i}^{(j,2)}\} \in E_t$ for all $j \in [3]$ and for all $t \in [3]$. In the second layer, we connect the extension and the core as follows.

- Edge $\{v_{c_i}^{(1,1)}, v_{c_i}^{(1,2)}\}$ forms a C_4 with edge $\{v_{c_i}^{(2)}, v_{c_i}^{(1)}\}$, that is, $\{v_{c_i}^{(2)}, v_{c_i}^{(1,2)}\} \in E_2$ and $\{v_{c_i}^{(1)}, v_{c_i}^{(1,1)}\} \in E_2$.
- Edge $\{v_{c_i}^{(2,1)}, v_{c_i}^{(2,2)}\}$ forms a C_4 with edge $\{v_{c_i}^{(2)}, v_{c_i}^{(3)}\}$, that is, $\{v_{c_i}^{(2)}, v_{c_i}^{(2,1)}\} \in E_2$ and $\{v_{c_i}^{(3)}, v_{c_i}^{(2,2)}\} \in E_2$.
- Edge $\{v_{c_i}^{(3,1)}, v_{c_i}^{(3,2)}\}$ forms a C_4 with edge $\{v_{c_i}^{(1)}, v_{c_i}^{(3)}\}$, that is, $\{v_{c_i}^{(1)}, v_{c_i}^{(3,2)}\} \in E_2$ and $\{v_{c_i}^{(3)}, v_{c_i}^{(3,1)}\} \in E_2$.

Lastly, we introduce nine auxiliary vertices that help to connect clause gadgets and variable gadgets. Let these vertices be called $v_{c_i}^{(j,1,1)}$, $v_{c_i}^{(j,1,2)}$, and $v_{c_i}^{(j,2,1)}$ for all $j \in [3]$. In the third layer, we connect the extension of the core and these auxiliary vertices in the following way. For all $j \in [3]$ we have that $\{v_{c_i}^{(j,1,1)}, v_{c_i}^{(j,1,2)}\} \in E_3$, $\{v_{c_i}^{(j,1,2)}, v_{c_i}^{(j,1)}\} \in E_3$, and $\{v_{c_i}^{(j,2,1)}, v_{c_i}^{(j,2)}\} \in E_3$.

Connection of variable and clause gadgets. The clause gadgets and variable gadgets are connected in the third layer. Let clause $c_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ with $i \in [m]$ have literals $\ell_{i,1}$, $\ell_{i,2}$, and $\ell_{i,3}$. Let $x_{i,j}$ with $i \in [m]$ and $j \in [3]$ be the variable of the jth literal in clause c_i. If $\ell_{i,j} = x_{i,j}$, then $\{v_{x_{i,j}}^{(2)}, v_{c_i}^{(j,1,1)}\} \in E_3$ and $\{v_{x_{i,j}}^{(3)}, v_{c_i}^{(j,2,1)}\} \in E_3$. If $\ell_{i,j} = \neg x_{i,j}$, then $\{v_{x_{i,j}}^{(1)}, v_{c_i}^{(j,1,1)}\} \in E_3$ and $\{v_{x_{i,j}}^{(2)}, v_{c_i}^{(j,2,1)}\} \in E_3$. This completes the construction. Recall that $\Delta = 2$ and $k = 2$.

Properties of $\mathcal{G}$. We can check that the underlying graph is 3-colorable: It is easy to see that we can color each variable gadget with three colors in the underlying graph. The same for each clause gadget (without the connecting auxiliary vertices). The auxiliary vertices can now be colored as follows. Vertices $v_{c_i}^{(j,2,1)}$ are connected to two

vertices with potentially different colors. Hence, we can use the third color for $v_{c_i}^{(j,2,1)}$. We can color $v_{c_i}^{(j,1,2)}$ with a color that is different from the color of $v_{c_i}^{(j,1)}$. Now $v_{c_i}^{(j,1,1)}$ is connected to two vertices with potentially different colors. Hence, we can use the third color for $v_{c_i}^{(j,1,1)}$.

To see that the underlying graph has constant maximum degree recall that every variable appears in exactly four clauses. Hence, the vertices $v_{x_i}^{(1)}$ have degree at most seven in the underlying graph. It is straightforward to check that all other vertices also have degree at most seven.

Lastly, we can easily verify that all layers are composed of small connected components (see also Figure 6.4). To see this, recall that every variable appears in exactly four clauses, hence in the third layer, each variable gadget is connected to four extensions of clause gadgets.

Correctness. It is easy to check that the reduction can be computed in polynomial time. It remains to show that $\mathcal{G}$ admits a proper sliding 2-window temporal 2-coloring if and only if ϕ is satisfiable.

($\Rightarrow$): Assume that we are given a satisfying assignment for ϕ. Then we construct a proper sliding 2-window temporal 2-coloring for $\mathcal{G}$ as follows. We start coloring the second layer and then show that we can color layers one and three in a way such that the complete coloring is a proper sliding 2-window temporal 2-coloring. If a variable x_i with $i \in [n]$ is set to true in the satisfying assignment, then we color the triangle of the corresponding variable gadget in a way that leaves only edge $\{v_{x_i}^{(1)}, v_{x_i}^{(2)}\}$ monochromatic. To be specific, assume (for the remainder of this paragraph) we have colors yellow and blue, we color vertices $v_{x_i}^{(1)}$ and $v_{x_i}^{(2)}$ in yellow and vertices $v_{x_i}^{(3)}$, $v_{x_i}^{(4)}$, $v_{x_i}^{(5)}$ in blue. If variable x_i is set to false in the satisfying assignment, then we color the triangle of the corresponding variable gadget in a way that leaves edge $\{v_{x_i}^{(2)}, v_{x_i}^{(3)}\}$ monochromatic. To be specific, we color vertices $v_{x_i}^{(2)}$ and $v_{x_i}^{(3)}$ in yellow and vertices $v_{x_i}^{(1)}$, $v_{x_i}^{(4)}$, $v_{x_i}^{(5)}$ in blue. For each clause c_i with $i \in [m]$ we choose one of its literals that satisfies the clause. Let the jth literal with $j \in [3]$ be a satisfying literal of clause c_i for the given assignment. Then we color the core of the corresponding clause gadget in a way that leaves edge $\{v_{c_i}^{(j)}, v_{c_i}^{(j \bmod 3+1)}\}$ monochromatic. Note that coloring the core uniquely determines how we have to color the extension of the core since the connecting edges are only present in the third layer and hence have to be properly colored. The auxiliary vertices can be colored arbitrarily.

Now we show how to color layer one. For each variable x_i with $i \in [n]$, we color $v_{x_i}^{(2)}$ in yellow and the remaining vertices of the corresponding gadget in blue. Note that this ensures that the edge which remains monochromatic in the second layer is properly colored in the first layer. For each clause c_i with $i \in [m]$ we color the core in

a way that ensures that the edge which remains monochromatic in the second layer is properly colored in the first layer. We properly color all edges of the extension and the auxiliary vertices arbitrarily. It is not hard to see that now the first Δ-window is properly colored.

Lastly, we show how to color the third layer. Note that for the variable gadgets, the coloring in layer two determines (up to renaming the colors) how to color the variable gadgets in the third layer. This also determines how to color the auxiliary vertices and the extension of the core in the third layer. This potentially leaves edges of the extension monochromatic. Note that in the second layer, all extension edges are properly colored except the one which, in the third layer, is connected to a variable that, in the given assignment, satisfies the clause. It is straightforward to check that in this case, this particular extension edge is properly colored in the third layer. Lastly, the core is colored in a way that ensures that the edge that is colored monochromatically in the second layer is colored properly in the third layer. It is easy to check that now the second Δ-window is also properly colored.

($\Leftarrow$): Assume we are given a proper sliding 2-window temporal 2-coloring for $\mathcal{G}$. Then we construct a satisfying assignment for ϕ in the following way: Note that in the second layer each variable gadget contains a triangle with exactly one monochromatic edge. The edge $\{v_{x_i}^{(1)}, v_{x_i}^{(3)}\}$ only exists in the second layer and hence is colored properly by any proper sliding 2-window temporal 2-coloring. This means that either edge $\{v_{x_i}^{(1)}, v_{x_i}^{(2)}\}$ or edge $\{v_{x_i}^{(2)}, v_{x_i}^{(3)}\}$ is colored monochromatically. If $\{v_{x_i}^{(1)}, v_{x_i}^{(2)}\}$ is colored monochromatically, then we set x_i to true, otherwise we set x_i to false. We claim that this yields a satisfying assignment for ϕ.

Assume for contradiction that it is not. Then there is a clause c_j that is not satisfied. Without loss of generality, let x_1, x_2, and x_3 be the variables appearing in c_j. Then in the third layer, the clause gadget of c_j is connected to the variable gadgets of x_1, x_2, and x_3. It is easy to check that in any proper sliding 2-window temporal 2-coloring, exactly one edge of the extension of any clause gadget is colored monochromatically in the second layer, hence this is also the case in the clause gadget of c_j. Without loss of generality, let the monochromatically colored (in the second layer) extension edge of the clause gadget of c_j be connected to the variable gadget of x_1 in the third layer. It is easy to check that for the sliding 2-window temporal 2-coloring to be proper, the edge of the variable gadget of x_1 that is connected to the clause gadget of c_j in the third layer needs to be colored properly in the second layer. By construction of $\mathcal{G}$ this is a contradiction to c_j not being satisfied by the constructed assignment.

Modification for a Larger Number of Colors. To modify this reduction for more colors we introduce new vertices and edges to all layers to "block" all colors except two

from being used. Formally, we do the following. Let $k > 2$. For each layer $i \in [3]$, we add $k-2$ fresh vertices $c_1^{(i)}, \ldots, c_{k-2}^{(i)}$ for each connected component C in that layer. The vertices $c_1^{(i)}, \ldots, c_{k-2}^{(i)}$ form a clique in layer i, and we connect them to all vertices in the connected component C. In all layers different from i the vertices $c_1^{(i)}, \ldots, c_{k-2}^{(i)}$ are isolated. All new edges exist in exactly one layer and hence have to be colored properly in this layer. It follows that the vertices $c_1^{(i)}, \ldots, c_{k-1}^{(i)}$ have to be colored with $k-2$ distinct colors and these colors then cannot be used to color any other vertex from the connected component in layer i.

The number of new vertices introduced by this modification is in $O(n \cdot k)$. It is easy to check that this increases the number of colors necessary to color the underlying graph by $k-2$. The maximum degree of the underlying graph after the modification is in $O(k)$ and the size of each connected component in each layer is increased by $k-2$. □

With small modifications to the reduction we get that Sliding Window Temporal Coloring remains hard under the following restrictions on the layers.

Corollary 6.6. Sliding Window Temporal Coloring *is NP-complete for all* $k \geq 2$,

- $\Delta \in O(k^2)$, *and* $\ell \geq \Delta + 1$ *even if every layer is a cluster graph, or*
- $\Delta \geq 3$, *and* $\ell \geq \Delta + 1$ *even if every layer has domination number one.*

Proof. Both modifications rely on the following construction. We can insert additional vertices and edges to each of the three layers of the reduction presented in the proof of Theorem 6.5. Between layers two and three we add sufficiently many new layers containing exclusively new edges, such that all new edges can be properly temporally colored at least once if Δ is increased by the number of new layers. Now all new edges can be colored properly in the newly inserted layers and the original construction of the reduction is not affected.

To get the first property for all layers, we can add all edges that transform each connected component into a clique to the three original layers. Since the components have size $O(k)$, we only add a number of new edges that is in $O(k^2)$ per component. Hence, we can $O(k^2)$ new layers each containing one new edge per component. The newly added layers are clearly cluster graphs, hence we get the result.

To get the second property, we add one new universal vertex and one new layer that contains all new edges. Clearly, now all layers have a dominating set of size one. □

We remark that Theorem 6.5 and Corollary 6.6 have interesting implications from a parameterized point of view. Parameterizing Sliding Window Temporal Coloring by the maximum degree of the underlying graph cannot yield fixed-parameter tractability unless P = NP, even if combined with k and ℓ. Furthermore, parameterizing Sliding Window Temporal Coloring by structural graph parameters of the layers that are constant if all connected components are constant-sized cannot yield fixed-parameter tractability unless P = NP, even if combined with k and ℓ.

The reduction presented in the proof of Theorem 6.5 also yields a running time lower bound assuming the Exponential Time Hypothesis (ETH) [IP01, IPZ01]. In the next section we will use this result to show that an algorithm we present presumably is asymptotically optimal if Δ is constant.

Corollary 6.7. Sliding Window Temporal Coloring *does not admit an* $f(k+\ell)^{o(|\mathcal{G}|)} \cdot |\mathcal{G}|^{f(k+\ell)}$*-time algorithm for any computable function* f *unless the ETH fails.*

Proof. First, note that any 3-SAT formula with m clauses can be transformed into an equisatisfiable Exact (3,4)-SAT formula with $O(m)$ clauses [Tov84]. The reduction presented in the proof of Theorem 6.5 produces an instance of Sliding Window Temporal Coloring with a temporal graph of size $|\mathcal{G}| \in O(m)$, $k = 2$, and $\ell = 3$. Hence an algorithm for Sliding Window Temporal Coloring with running time $f(k+\ell)^{o(|\mathcal{G}|)} \cdot |\mathcal{G}|^{f(k+\ell)}$ for some computable function f would imply the existence of a $2^{o(m)}$-time algorithm for 3-SAT. This is a contradiction to the ETH [IP01, IPZ01]. □

6.4.2 An Exponential-Time Algorithm for Sliding Window Temporal Coloring

In the following we give an exponential-time algorithm for Sliding Window Temporal Coloring that, if Δ is constant, asymptotically matches the running time lower bound given in Corollary 6.7 assuming the ETH to hold.

We start with an auxiliary technical observation that, intuitively, allows us to combine partial colorings if they agree on their "overlap", that is, time intervals of the temporal graph that are colored by both partial colorings, and if the overlap is sufficiently large.

Observation 6.8. *Let* $(\mathcal{G}_1 = (V, E_1, E_2, \ldots, E_i), k, \Delta)$ *and* $(\mathcal{G}_2 = (V, E_j, E_{j+1}, \ldots, E_\ell), k, \Delta)$ *be two instances of* Sliding Window Temporal Coloring *with* $j+\Delta-1 \leq i \leq \ell$*. Let* $\Upsilon^{(1)}$ *and* $\Upsilon^{(2)}$ *be sliding* Δ*-window temporal colorings for* $\mathcal{G}_1$ *and* $\mathcal{G}_2$*, respectively, that use the same* k *colors and with the property that for all* $v \in V$ *and for all* $j \leq i^\star \leq i$ *we have that* $\Upsilon^{(1)}(v, i^\star) = \Upsilon^{(2)}(v, i^\star)$.

Then we have that $\Upsilon^{(1)}$ *and* $\Upsilon^{(2)}$ *are proper sliding* Δ*-window temporal colorings if and only if* $(\Upsilon^{(1)}_1, \Upsilon^{(1)}_2, \ldots, \Upsilon^{(1)}_i, \Upsilon^{(2)}_{i+1}, \ldots, \Upsilon^{(2)}_\ell)$ *is a proper sliding* Δ*-window temporal coloring for* $\mathcal{G} = (V, E_1, E_2, \ldots, E_\ell)$.

Proof. It is easy to see that if $(\Upsilon_1^{(1)}, \Upsilon_2^{(1)}, \ldots, \Upsilon_i^{(1)}, \Upsilon_{i+1}^{(2)}, \ldots, \Upsilon_\ell^{(2)})$ is a proper sliding Δ-window temporal coloring for $\mathcal{G} = (V, E_1, E_2, \ldots, E_\ell)$, then we have that $\Upsilon^{(1)}$ and $\Upsilon^{(2)}$ are proper sliding Δ-window temporal colorings for $(\mathcal{G}_1 = (V, E_1, E_2, \ldots, E_i), k, \Delta)$ and $(\mathcal{G}_2 = (V, E_j, E_{j+1}, \ldots, E_\ell), k, \Delta)$, respectively. For the other direction, assume for contradiction that $(\Upsilon_1^{(1)}, \Upsilon_2^{(1)}, \ldots, \Upsilon_i^{(1)}, \Upsilon_{i+1}^{(2)}, \ldots, \Upsilon_\ell^{(2)})$ is *not* a proper sliding Δ-window temporal coloring for $\mathcal{G} = (V, E_1, E_2, \ldots, E_\ell)$. Then there is a Δ-window W_t^Δ for some t and an edge $e \in E_{t'}$ for some $t' \in W_t^\Delta$ that is never properly colored in W_t^Δ. However, it is easy to check that W_t^Δ is completely contained in $[i]$ or $[j, \ell]$ (because of the bounds on i and j) and hence $\Upsilon^{(1)}$ or $\Upsilon^{(2)}$ color the whole Δ-window W_t^Δ. Since, by assumption, both $\Upsilon^{(1)}$ and $\Upsilon^{(2)}$ are proper sliding Δ-window temporal colorings, there being an edge that exists in W_t^Δ and is not properly colored is a contradiction. □

Now we are ready to describe an exponential-time algorithm for Sliding Window Temporal Coloring. The main idea is to enumerate all partial proper sliding Δ-window temporal colorings for time windows of size 2Δ and then to check whether we can combine them to a proper sliding Δ-window temporal coloring for the whole temporal graph using Observation 6.8.

Theorem 6.9. Sliding Window Temporal Coloring *can be solved in* $O(k^{4\Delta \cdot |V|} \cdot \ell)$ *time.*

Proof. Let $(\mathcal{G} = (V, (E_i)_{i \in [\ell]}), k, \Delta)$ be an input instance for Sliding Window Temporal Coloring. For the sake of simplicity, we assume that ℓ is divisible by Δ. If this is not the case, we can "mirror" the last layers: we repeat layers $\{\ell - (\ell \bmod \Delta) - 1, \ldots, \ell - 1\}$ in reverse order after the ℓth layer. We give an algorithm for this problem that works as follows:

1. For 2Δ-windows $W_{i\Delta+1}^{2\Delta} = [i\Delta + 1, (i+2)\Delta]$ for $i \in \{0, 1, \ldots, \ell/\Delta - 2\}$, enumerate all partial proper sliding Δ-window temporal colorings $\Upsilon_{W_{i\Delta+1}^{2\Delta}}^{(j)}$ that use at most k fixed colors, where each trivial layer is colored in some fixed but arbitrary way[13].

2. Create a *directed acyclic graph* (DAG) with all $\Upsilon_{W_{i\Delta+1}^{2\Delta}}^{(j)}$ as vertices and connect $\Upsilon_{W_{i\Delta+1}^{2\Delta}}^{(j)}$ and $\Upsilon_{W_{(i+1)\Delta+1}^{2\Delta}}^{(j')}$ with a directed arc if the two proper sliding Δ-window temporal colorings agree on the overlapping part.

[13] This is an important "trick" that will allow us to use this algorithm for the FPT result in Theorem 6.10.

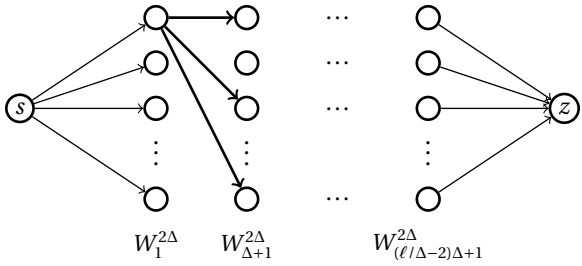


Figure 6.5: Illustration of the DAG constructed in the algorithm described in the proof of Theorem 6.9. Each vertex in each column of vertices corresponds to a partial proper sliding Δ-window temporal coloring for the Δ-window written at the bottom of the column. Thick arcs are only included in the DAG if the two connected partial proper sliding Δ-window temporal colorings agree on the overlapping part.

3. Create a source vertex s and connect it to all $\Upsilon^{(j)}_{W_1^{2\Delta}}$ with a directed arc and create a sink vertex z and add a directed arc from all $\Upsilon^{(j)}_{W^{2\Delta}_{(\ell/\Delta-2)\Delta+1}}$ to it.

4. If there is a path from s to z, then answer YES, otherwise NO.

The constructed DAG is visualized in Figure 6.5.

Correctness. We now show that the above described algorithm is correct.

($\Rightarrow$): Assume that we are given a proper sliding Δ-window temporal coloring Υ that uses at most k colors for $\mathscr{G}$. Without loss of generality, let the k colors be the same fixed k colors we use in the algorithm. If $\mathscr{G}$ contains trivial layers, we assume without loss of generality that Υ colors them in the same fixed but arbitrary way as we do in the algorithm. Then for 2Δ-windows $W^{2\Delta}_{i\Delta+1} = [i\Delta+1,(i+2)\Delta]$ for $i \in \{0,1,\ldots,\ell/\Delta-2\}$, the partial coloring of $W^{2\Delta}_{i\Delta+1}$ that agrees with Υ appears in the constructed DAG, since by assumption it is proper and we enumerate all of them. Now for any two 2Δ-windows $W^{2\Delta}_{i\Delta+1}$ and $W^{2\Delta}_{(i+1)\Delta+1}$ we obviously have that if we color them with Υ the overlapping part is colored in the same way. Hence the vertices corresponding to the implied partial coloring for these two 2Δ-windows are connected. Following these connections, we can see that we find a path from s to z in the constructed DAG.

($\Leftarrow$): If there is a path from s to z in the constructed DAG, then, by Observation 6.8, we can combine the partial proper sliding Δ-window temporal coloring corresponding to the vertices visited by the path since, by construction, they overlap for Δ time

steps and agree on how to color the vertices in the overlapping part. This given us a proper sliding Δ-window temporal coloring for the whole graph.

Running Time. The running time is dominated by checking whether s and z are connected in the last step of the algorithm. This can be done for example by a breadth-first-search on the constructed DAG. The DAG has at most $k^{2\Delta\cdot|V|}\cdot\ell$ vertices and at most $k^{4\Delta\cdot|V|}\cdot\ell$ edges. □

6.4.3 An FPT-Algorithm for Sliding Window Temporal Coloring

In this section, we show how to extend the algorithm presented in Theorem 6.9 to achieve linear-time fixed-parameter tractability with respect to the number of vertices. The main idea is to reduce the number of non-trivial layers in each Δ-window. However, the procedure we describe only guarantees a very large upper bound on the number of non-trivial layers in each Δ-window. Hence, the following result is only of classification nature.

Theorem 6.10. SLIDING WINDOW TEMPORAL COLORING *can be solved in* $2^{O(2^{|V|^2})}\cdot\ell$ *time.*

Proof. We present a preprocessing step to reduce the number of non-trivial layers in any Δ-window and then use the algorithm of Theorem 6.9 to solve the problem.

The data reduction rule is based on the observation that if some layer appears at least $|V|^2$ times in a Δ-window, then the edges of this layer can be properly colored with two colors within the Δ-window. In other words, all but $|V|^2$ copies of the layer in the Δ-window are redundant for optimal coloring and each of them could be replaced by the trivial layer. When implementing this idea one should take care to guarantee that replacing a layer by the trivial one does not reduce the number of copies of the layer in other Δ-windows which contain at most $|V|^2$ copies of the layer.

Formally, the data reduction rule is as follows. Since the number of different layers is at most $2^{\binom{|V|}{2}}\leq 2^{|V|^2}$, by the pigeonhole principle if $\Delta > 2\cdot 2^{|V|^2}\cdot|V|^2$, then in every Δ-window there exists a layer that appears more than $2|V|^2$ times in that Δ-window. For every such layer that contains at least one edge, we replace one of its "middle" copies, that is, one that has at least $|V|^2$ copies appearing earlier and $|V|^2$ copies that appear later in the Δ-window, by a trivial layer. This data reduction rule guarantees that every Δ-window that contains the modified layer also contains at least $|V|^2$ copies of the original layer appearing either earlier or later in the Δ-window.

The data reduction rule can be applied exhaustively by linearly sweeping over all Δ-windows once in the following way. For each different graph (layer) we store a list of occurrences and update these lists every time we move the Δ-window by one.

Having these lists, it is straightforward to count the occurrences and replace the middle ones by trivial layers. When we move the Δ-window, we just have to update two lists: the one of the graph that enters the Δ-window and the one of the graph that leaves. This requires a lookup table of size $2^{\binom{|V|}{2}} \leq 2^{|V|^2}$ but takes only time linear in ℓ. Note that after this procedure, every Δ-window contains at most $2 \cdot 2^{|V|^2} \cdot |V|^2$ non-trivial layers.

Now we apply the algorithm of Theorem 6.9. Note that after the data reduction step the number of *non-trivial* layers in every Δ-window depends only on $|V|$. Furthermore, since we can assume that $k \leq |V|$, the number of colorings that are enumerated in Step 1 of the algorithm in Theorem 6.9 is in $2^{O(2^{|V|^2})}$. This completes the proof. □

We complement the fixed-parameter tractability result of Theorem 6.10 with the following proposition, in which we exclude the possibility of a polynomial-sized kernel for Sliding Window Temporal Coloring when parameterized by the number $|V|$ of vertices unless NP $\subseteq$ coNP/poly. This stands in contrast to the existence of a polynomial kernel for Temporal Coloring when parameterized by $|V|$ [MMZ19].

Proposition 6.11. Sliding Window Temporal Coloring *parameterized by the number $|V|$ of vertices does not admit a polynomial kernel for all $\Delta \geq 2$ and $k \geq 2$ unless NP $\subseteq$ coNP/poly.*

Proof. We provide an AND-cross-composition (for a definition see Section 2.3) from Exact (3,4)-SAT [Tov84]. Recall that in Exact (3,4)-SAT we are asked to decide whether a given Boolean formula ϕ is satisfiable and ϕ is in conjunctive normal form where every clause has exactly three distinct literals and every variable appears in exactly four clauses. Intuitively, we can just string together instances produced by the reduction we presented in the proof of Theorem 6.5 in the time axis with some extra layers in between such that the large instance admits a proper sliding Δ-window temporal coloring if and only if all original instances are YES-instances.

We define an equivalence relation R as follows: Two instances ϕ and ψ are equivalent under R if and only if the number of variables and the number of clauses is the same in both formulas. Clearly, R is a polynomial equivalence relation.

Now let $\phi_1, \ldots, \phi_n$ be R-equivalent instances of Exact (3,4)-SAT. We arbitrarily number all variables and clauses of all formulas. For each ϕ_i with $i \in [n]$ we construct an instance of Sliding Window Temporal Coloring as defined in the proof of Theorem 6.5 (for an illustration see Figure 6.4) with the only difference that we add a fourth and fifth layer both of which are copies of the first layer (Figure 6.4a). Now we put all constructed temporal graphs next to each other in temporal order, that is, if $\mathcal{G}^{(i)} = (V, E_1^{(i)}, E_2^{(i)}, \ldots, E_5^{(i)})$ is the graph constructed for ϕ_i, then the overall

temporal graph is $\mathcal{G} = (V, E_1^{(1)}, E_2^{(1)}, \ldots, E_5^{(1)}, E_1^{(2)}, E_2^{(2)}, \ldots, E_5^{(2)}, \ldots, E_1^{(n)}, E_2^{(n)}, \ldots, E_5^{(n)})$. Here, the vertex set stays the same. We identify the vertices with their names according to the numbering of the variables and clauses of the formulas. Further, we set $\Delta = 2$ and $k = 2$.

This instance can be constructed in polynomial time and the number of vertices is linearly upper-bounded in the size of the formulas, hence $|V|$ is polynomially upper-bounded by the maximum size of an input instance. Furthermore, it is easy to check that the two extra copies of the first layer in the construction (Figure 6.4a) allow to go from an arbitrary proper coloring of layer $G_4^{(i)} = (V, E_4^{(i)})$ to $G_1^{(i+1)} = (V, E_1^{(i+1)})$ for any $i \in [n-1]$. It follows from the proof of Theorem 6.5 that the constructed Sliding Window Temporal Coloring instance is a YES-instance if and only if for every $i \in [n]$ formula ϕ_i is satisfiable.

Since Exact (3,4)-SAT is NP-hard [Tov84] and we AND-cross-composed it into Sliding Window Temporal Coloring with $\Delta = 2$ and $k = 2$ parameterized by $|V|$, the result follows. □

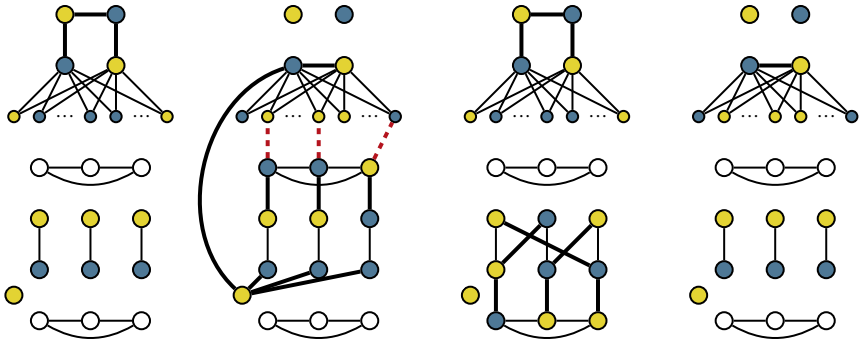
6.4.4 Structural Graph Parameters and Approximation

In this section, we investigate the possibility to improve the fixed-parameter tractability result of Theorem 6.10 by replacing the parameter $|V|$ with a smaller parameter. We answer this negatively by showing that Sliding Window Temporal Coloring remains NP-complete even if the underlying graph has a vertex cover number in $O(k)$, which is a fairly large structural parameter.

Theorem 6.12. Sliding Window Temporal Coloring *is NP-complete for all* $k \geq 2$, *even if* $\Delta = 2$ *and the vertex cover number of the underlying graph is in* $O(k)$.

Proof. We present a polynomial-time reduction from Monotone Exactly 1-in-3 SAT [GJ79, Sch78] to Sliding Window Temporal Coloring with $k = 2$ and $\Delta = 2$. The reduction can be easily modified to a larger number of colors, we explain how to do this at the end of the proof. In Monotone Exactly 1-in-3 SAT we are given a collection of triples (clauses) of variables and the task is to determine whether there is an assignment of truth values to variables such that each clause contains exactly one variable that is set to true. Given an instance I of Monotone Exactly 1-in-3 SAT with n variables and m clauses, we construct a temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ with $\ell = 4m$ layers in the following way. The construction is visualized in Figure 6.6.

Construction. In the construction, we classify the layers of the constructed temporal graph by the remainders of their time steps when divided by four. This gives us *type 1*, *type 2*, *type 3*, and *type 4* layers, where type 4 layers are the ones with a time step that



(a) Type 1 layer. **(b)** Type 2 layer. **(c)** Type 3 layer. **(d)** Type 4 layer.

Figure 6.6: Illustration of the reduction from MONOTONE EXACTLY 1-IN-3 SAT to SLIDING WINDOW TEMPORAL COLORING of the proof of Theorem 6.12. The vertex numbering in the description of the construction corresponds to a row-wise numbering from top-left to bottom-right. The first two rows correspond to vertices u_1 to u_4. The third row corresponds to vertices v_1 to v_n. The remaining rows correspond to vertices w_1 to w_{13}. Thin edges appear in all layers. Thick edges never appear consecutively and hence need to be colored properly. Red dashed edges correspond to clauses. The colors of the vertices correspond to the proper sliding Δ-window temporal coloring constructed in the proof of Theorem 6.12.

is divisible by four and the other type numbers correspond to the remainders of the time steps. We start by adding four vertices u_1, u_2, u_3, and u_4 to G. In layers of type 1 or type 3 we add edges $\{u_1, u_2\}$, $\{u_1, u_3\}$, and $\{u_2, u_4\}$. In layers of type 2 or type 4 we add edge $\{u_3, u_4\}$. For each variable x_i we add a vertex v_i. We connect each of u_3 and u_4 to all v_i in all layers. Next, we add 13 further vertices $w_1, w_2, \ldots, w_{13}$ to V. In all layers we pairwise connect w_1, w_2, and w_3, pairwise connect w_{11}, w_{12}, and w_{13}, and add edges $\{w_4, w_7\}$, $\{w_5, w_8\}$, and $\{w_6, w_9\}$. In layers of type 2 we add edges $\{w_1, w_4\}$, $\{w_2, w_5\}$, $\{w_3, w_6\}$, $\{u_3, w_{10}\}$, $\{w_7, w_{10}\}$, $\{w_8, w_{10}\}$, and $\{w_9, w_{10}\}$ (see Figure 6.6b). In layers of type 3 we add edges $\{w_4, w_9\}$, $\{w_5, w_7\}$, $\{w_6, w_8\}$, $\{w_7, w_{11}\}$, $\{w_8, w_{12}\}$, and $\{w_9, w_{13}\}$ (see Figure 6.6c). Lastly, let x_{i_1}, x_{i_2}, and x_{i_3} be the three variables contained in clause c_j. Then we add edges $\{v_{i_1}, w_1\}$, $\{v_{i_2}, w_2\}$, and $\{v_{i_3}, w_3\}$ in layer $4j-2$ (see red edges in Figure 6.6b). Note that layer $4j-2$ has type 2.

Correctness. It is easy to check that this can be done in polynomial time. Note that vertices $u_1, u_2, u_3, u_4, w_1, w_2, \ldots, w_{13}$ form a vertex cover in $G_\downarrow$. We are ready now to prove that the MONOTONE EXACTLY 1-IN-3 SAT instance I is a YES-instance if and

only if $\mathcal{G}$ admits a proper sliding 2-window temporal 2-coloring.

($\Rightarrow$): Assume we are given a YES-instance of MONOTONE EXACTLY 1-IN-3 SAT with a satisfying assignment. We show that the constructed instance of SLIDING WINDOW TEMPORAL COLORING is also a YES-instance by presenting a proper sliding Δ-window temporal coloring with two colors. Let blue and yellow be the two colors we use. We always color u_1 and u_4 yellow and u_2 and u_3 blue. If variable x_i is set to true in the satisfying assignment, then we color v_i yellow in layers of type 1 and 3 and blue in layers of type 2 and 4. If variable x_i is set to false in the satisfying assignment, then we color v_i yellow in layers of type 2 and 4 and blue in layers of type 1 and 3. Vertex w_{10} is always colored yellow.

Let clause c_j be satisfied by its sth variable (note that $s \in [3]$). We describe how to color vertices $w_1, \ldots, w_9$ in layer $4 \cdot j - 2$. Vertex w_s is colored yellow. Vertices in $\{w_1, w_2, w_3\} \setminus \{w_s\}$ are colored blue. Vertex w_{s+3} is colored blue. Vertices in $\{w_4, w_5, w_6\} \setminus \{w_{s+3}\}$ are colored yellow. Vertices w_7, w_8, and w_9 are colored blue. We further describe how to color vertices $w_4, \ldots, w_{13}$ in layer $4 \cdot j - 1$. Note that w_{10} is already colored yellow.

- We color vertex $w_{((s+1) \bmod 3)+4}$ blue and vertices in $\{w_4, w_5, w_6\} \setminus \{w_{((s+1) \bmod 3)+4}\}$ yellow.
- We color vertex $w_{(s \bmod 3)+7}$ yellow and vertices in $\{w_7, w_8, w_9\} \setminus \{w_{(s \bmod 3)+7}\}$ blue.
- We color vertex $w_{(s \bmod 3)+11}$ blue and vertices in $\{w_{11}, w_{12}, w_{13}\} \setminus \{w_{(s \bmod 3)+11}\}$ yellow.

In all layers of type 1 and 4 we color vertices w_4, w_5, and w_6 yellow and vertices w_7, w_8, and w_9 blue. The coloring scheme so far is depicted in Figure 6.6. Note that the colors of some vertices in some layers are not specified yet. These are the white vertices in Figure 6.6. All these vertices belong to triangles, hence we can color them in a way that each triangle has one monochromatic edge. We choose as the edge that should remain monochromatic an edge that is properly colored in both adjacent layers. Such an edge always exists since all these triangles are also triangles in the adjacent layers and a triangle is never colored completely monochromatically.

($\Leftarrow$): Assume that the constructed instance of SLIDING WINDOW TEMPORAL COLORING is a YES-instance and that we have a proper sliding Δ-window temporal coloring with two colors. We show that the given instance of MONOTONE EXACTLY 1-IN-3 SAT is also a YES-instance by constructing a satisfying assignment. We claim that the following yields a satisfying assignment. For every variable x_i, if edge $\{u_3, v_i\}$ is colored properly in the first layer, then we set x_i to true, otherwise we set x_i to false.

First we argue that if an edge $\{u_3, v_i\}$ is colored properly in the first layer for some $i \in [n]$, then it is also colored properly in every odd layer (that is, every layer of type 1 and 3). Furthermore, the edge is colored monochromatically in every even layer (that is, every layer of type 2 and 4). Analogously, if an edge $\{u_3, v_i\}$ is colored monochromatically in the first layer for some $i \in [n]$, then it is also colored monochromatically in every odd layer and colored properly in every even layer. This follows from an easily verifiable fact that in every proper sliding Δ-window temporal coloring vertex u_3 is colored different from vertex u_4 in every layer. It follows that if an edge $\{u_3, v_i\}$ is colored monochromatically in a layer t for some $i \in [n]$ and $t \in [\ell - 1]$, then $\{u_3, v_i\}$ needs to be colored properly in layer $t+1$ meaning that $\{u_4, v_i\}$ is colored monochromatically in layer $t+1$. Symmetrically, if an edge $\{u_4, v_i\}$ is colored monochromatically in a layer t for some $i \in [n]$ and $t \in [\ell - 1]$, then $\{u_4, v_i\}$ needs to be colored properly in layer $t+1$ meaning that $\{u_3, v_i\}$ is colored monochromatically in layer $t+1$.

Now we are ready to argue that each clause of the MONOTONE EXACTLY 1-IN-3 SAT instance is satisfied. To see this we first take a look at layers of type 3. Note that the triangle consisting of vertices w_{11}, w_{12}, and w_{13} has exactly one monochromatic edge. It cannot have three since not all three edges can be colored properly in the adjacent layers. This means that exactly two out of the three vertices w_{11}, w_{12}, and w_{13} have the same color. It follows that exactly two out of the three vertices w_7, w_8, and w_9 have the same color, since the edges $\{w_7, w_{11}\}$, $\{w_8, w_{12}\}$, and $\{w_9, w_{13}\}$ need to be colored properly. It is easy to check that this implies that exactly two out of the three edges $\{w_4, w_7\}$, $\{w_5, w_8\}$, and $\{w_6, w_9\}$ are colored monochromatically, since edges $\{w_4, w_9\}$, $\{w_5, w_7\}$, and $\{w_6, w_8\}$ need to be colored properly.

Now we take a look at layers of type 2. From the last paragraph follows that at most one out of the three edges $\{w_4, w_7\}$, $\{w_5, w_8\}$, and $\{w_6, w_9\}$ is colored monochromatically. Since edges $\{u_3, w_{10}\}$, $\{w_7, w_{10}\}$, $\{w_8, w_{10}\}$, and $\{w_9, w_{10}\}$ need to be colored properly, we have that vertices w_7, w_8, and w_9 have the same color as vertex u_3. It follows that at most one of the vertices w_4, w_5, and w_6 is colored in the same color as u_3. Since vertices w_1, w_2, and w_3 form a triangle and edges $\{w_1, w_4\}$, $\{w_2, w_5\}$, and $\{w_3, w_6\}$ need to be colored properly, it follows that exactly one out of the three vertices w_1, w_2, and w_3 is colored differently from u_3. Recall that w_1, w_2, and w_3 are connected to vertices v_{i_1}, v_{i_2}, and v_{i_3} corresponding to the three variables x_{i_1}, x_{i_2}, and x_{i_3} that are contained in the clause that corresponds to the layer. The connecting edges need to be colored properly. Consequently, exactly one of the edges $\{u_3, v_{i_1}\}$, $\{u_3, v_{i_2}\}$, and $\{u_3, v_{i_3}\}$ is colored monochromatically and the other two are colored properly. It follows that in the first layer, exactly one of the edges $\{u_3, v_{i_1}\}$, $\{u_3, v_{i_2}\}$, and $\{u_3, v_{i_3}\}$ is colored properly and the other two are colored monochromatically.

This means we set exactly one of the three variables to true and the clause is satisfied.

Modification for a Larger Number of Colors. To modify this reduction for more colors we introduce new vertices and edges to the layers to "block" all colors except two from being used. Formally, we do the following. Let $k > 2$. We add $2k-4$ fresh vertices $c_1^{(1)},\dots,c_{k-2}^{(1)}$ and $c_1^{(2)},\dots,c_{k-2}^{(2)}$. In each layer of type 1 or 3 the vertices $c_1^{(1)},\dots,c_{k-2}^{(1)}$ form a clique and we connect them to all other vertices. In each layer of type 2 or 4 the vertices $c_1^{(2)},\dots,c_{k-2}^{(2)}$ form a clique and connect them to all other vertices. All new edges exist exactly once during any Δ-window for $\Delta = 2$ and hence have to be colored properly in every layer in which they appear. It follows that all vertices $c_1^{(i)},\dots,c_{k-1}^{(i)}$ have to be colored with $k-2$ distinct colors and these colors then cannot be used to color any other vertex.

The number of new vertices introduced by this modification is in $O(k)$ and we can simply add all of them to the vertex cover of the underlying graph. □

Finally, we consider a canonical optimization version of Sliding Window Temporal Coloring, which we call Minimum Sliding Window Temporal Coloring, where the goal is to minimize the number of colors k. Using Theorem 6.10, we provide an FPT-approximation algorithm with an additive error of one where the parameter is the vertex cover number of the underlying graph. Considering that we cannot hope for an exact FPT-algorithm for parameter "vertex cover number of the underlying graph" unless P = NP (Theorem 6.12), this is the best we can get from a classification standpoint.

Theorem 6.13. Minimum Sliding Window Temporal Coloring *admits an approximation algorithm with a running time in* $2^{O(2^{\mathrm{vc}_\downarrow^2})} \cdot \ell$ *and an additive error of one, where* $\mathrm{vc}_\downarrow$ *is the vertex cover number of the underlying graph.*

Proof. Let $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$ be the input temporal graph. First, we compute a minimum vertex cover $S \subseteq V$ of the underlying graph $G_\downarrow$. Let the size of this vertex cover be $\mathrm{vc}_\downarrow = |S|$. Note that this can be done in $O(2^{\mathrm{vc}_\downarrow} \cdot (|V| + |E(G_\downarrow)|)$ time [BG93, DF95]. We use the algorithm of Theorem 6.10 to compute the size of a minimum proper sliding Δ-window temporal coloring for the temporal graph $\mathcal{G}[S]$ induced by the vertex cover vertices[14]. By Theorem 6.10 this computation takes $2^{O(2^{\mathrm{vc}_\downarrow^2})} \cdot \ell$ time and the number of colors used is clearly a lower bound for the minimum number of

[14] Note that the algorithm presented in Theorem 6.10 solves the decision version of Sliding Window Temporal Coloring while we want to solve the minimization problem here. This can be done by trying out all values for the number k of colors between one and the size of the vertex cover of the underlying graph.

colors necessary to properly color the whole temporal graph. We color the remaining vertices with a fresh color. This clearly gives a proper sliding Δ-window temporal coloring for the whole temporal graph that uses at most one extra color compared to the optimum. □

6.5 Conclusion

In this chapter we introduced and studied two natural temporal extensions of the classic graph coloring problem, called Temporal Coloring and Sliding Window Temporal Coloring, where Temporal Coloring is the special case of Sliding Window Temporal Coloring, where the sliding window size equals the lifetime of the input temporal graph. For both variants we showed that they are NP-complete even under severe restrictions, in particular even if the number of colors is two, which stands in stark contrast to the static case, where the problem of coloring a graph with two colors is polynomial-time solvable.

On the positive side, we provided a linear-time FPT-algorithm for parameter "number $|V|$ of vertices" and a linear-time FPT-approximation algorithm for parameter "vertex cover number of the underlying graph" with an additive error of one. We leave as an open question whether for the latter we can replace vertex cover number by a structurally smaller parameter. However, due to their high space requirements these algorithms might not be very practical. A future research direction would be to investigate whether there are alternative algorithms yielding these tractability results that only require polynomial space.

There are several natural extensions of our problem that one could consider in future work. Considering our motivating example of mobile agents, it would be reasonable to assume that agents do not want to change a channel too frequently. In our model, this would translate to imposing a restriction on the number of color reassignments per vertex, or imposing a minimum time period that each vertex has to wait (after a color change) before it can change colors again. We remark that restricting the number of vertices that may change their color when going from one time step to the next (which would be a somewhat similar condition as used in "multistage" problems [Bam+18, Flu+19, GTW14]) presumably does not simplify the problem, since in the reduction of the proof of Theorem 6.12 this number is constant (for a constant number of colors k).

CHAPTER 7

Temporal Cliques and s-Plexes

One of the most fundamental problems in (social) network analysis is community detection, and one of the most basic primitives to model a community is a clique: a set of vertices that are all pairwise connected in the network. Addressing the problem of finding communities in temporal networks, Viard, Latapy, and Magnien [VLM16] introduced Δ-cliques as a natural temporal version of cliques. In this chapter, we show how to adapt the well-known Bron-Kerbosch algorithm [BK73] to enumerate Δ-cliques and temporal s-plexes (Δ-s-plexes), a temporal version of a popular clique relaxation.

We define a Δ-s-plex as a set of vertices and a time interval, where during this time interval each vertex has in each consecutive Δ time steps at least one edge to all but at most $s-1$ vertices in the chosen set of vertices. We develop a recursive algorithm for enumerating all maximal Δ-s-plexes. To analyze its running time, we introduce a temporal adaptation of the degeneracy of a graph, a popular measure for sparseness, which we call Δ-slice degeneracy. As our main result, we show that enumerating Δ-s-plexes is fixed-parameter tractable when parameterized by the Δ-slice degeneracy of the input temporal graph. We remark that our algorithm has been shown to perform very well in practice [Ben+19].

This chapter is based on a series of papers investigating the computational complexity of enumerating temporal cliques and temporal k-plexes [Ben+18, Ben+19, Him+16, Him+17].

7.1 Introduction

Network analysis is one of the main pillars of data science. Focusing on networks that are modeled by undirected graphs, a fundamental primitive is the identification of complete subgraphs, that is, cliques. This is particularly true in the context of detecting communities in social networks. Finding a maximum-cardinality clique in a graph is a classic NP-complete problem [Kar72], so exponential worst-case running time seems unavoidable. Moreover, often one wants to solve the more general task of not only finding one maximum-cardinality clique but to enumerate *all maximal* cliques. Their number can be exponential in the graph size. The famous Bron-Kerbosch algorithm ("Algorithm 457" in *Communications of the ACM, 1973*, [BK73])

addresses this task and still today forms the basis for some of the best (practical) algorithms to enumerate all maximal cliques in static graphs [Con+18, ELS13].

Cliques as a mathematical model, however, are often too restrictive for real-world applications, where some edges in communities might not exist because of errors in measurements or application-specific reasons. To mitigate this issue, the clique concept has seen several relaxations. In this chapter we focus on a popular degree-based relaxation of cliques known as s-plexes [SF78]. In an s-plex, every vertex must be adjacent to all but at most $s-1$ vertices in the s-plex (excluding itself). A 1-plex is a clique and in a 2-plex every vertex can have a missing edge to one other vertex in the 2-plex. One can use s-plexes also as a tool for link-prediction, as the missing edges are probably good candidates for missing links in social networks: It has been observed that friends of friends tend to become friends themselves [MZZ16].

Previous work on s-plexes uses *static* graph models [BBH11, BCK15, Con+17, Con+18, Guo+10, MH12, MNS12, PYB13, SF78, WP07, Xia+17]. However, to realistically model many real-world phenomena in social and other network structures, one has to take into account the dynamics of the modeled system of interactions between entities, which are captured nicely by temporal graphs. Indeed, as Nicosia et al. [Nic+13] pointed out, in many real-world systems the interactions among entities are rarely persistent over time and the non-temporal interpretation is an "oversimplifying approximation".

The generalization of a clique to the temporal setting that we study is called Δ-*clique* and was introduced by Viard, Latapy, and Magnien [VLM16]. Intuitively, being in a Δ-clique means to be regularly in contact with all other entities in this Δ-clique. In slightly more formal terms, each pair of vertices in the Δ-clique has to be in contact within at least every Δ time steps. We extend this concept to Δ-s-*plexes* by allowing each vertex to have up to $s-1$ missing edges during each interval of Δ consecutive time steps. It is easy to see that a Δ-1-plex is a Δ-clique. Fully formal definitions are given in Section 7.2. We present an adaption of the framework of Bron and Kerbosch [BK73] to temporal graphs and present an algorithm to enumerate Δ-cliques and Δ-s-plexes. To the best of our knowledge, we are the first to investigate the enumeration of s-plexes in temporal graphs.

7.1.1 Related Work

Finding maximum cliques (or cliques of a certain minimum cardinality) is a classic NP-complete problem [Kar72]. The enumeration of maximal cliques in static graphs is subject of many different algorithmic approaches (sometimes also exploiting specific properties such as the "degree of isolation" of the cliques searched for) [BK73, ELS13, Hüf+09, II09, Kom+09, TTT06].

The concept of s-plexes was introduced by Seidman and Foster [SF78]. There has been extensive research on maximum s-plex detection [BBH11, MH12, MNS12, Xia+17] and s-plex enumeration [BCK15, Con+17, Con+18, WP07] in static graphs. To find maximum s-plexes, there are several combinatorial branch-and-cut approaches [MH12, MNS12, Xia+17] as well as ILP-based algorithms [BBH11]. The Bron-Kerbosch algorithm [BK73] has been adapted to enumerate s-plexes in static graphs [Con+18, WP07], but there are also enumeration algorithms based on other approaches [BCK15, Con+17]. To the best of our knowledge, the currently asymptotically fastest algorithm for finding a maximum-cardinality s-plex in a static graph is due to Xiao et al. [Xia+17] and the asymptotically fastest algorithm for listing all maximal s-plexes in a static graph is due to Berlowitz, Cohen, and Kimelfeld [BCK15].

There are several other clique relaxations. Typically, the corresponding decision problems are NP-complete. For more details on different clique relaxations we refer to Pattillo, Youssef, and Butenko [PYB13].

The problem of finding Δ-cliques in temporal graphs was introduced and motivated by the study of Viard, Latapy, and Magnien [VLM16] who enumerated contact patterns among high-school students [BF14]. There has been intensive recent research on clique enumeration in temporal graphs [BP19, Qin+19, VLM16, VML18]. Recently, also the concept of isolation has be transferred to the temporal setting [MNR19].

7.1.2 Our Contributions and Organization of the Chapter

We formally introduce and define Δ-s-plexes (Definition 7.6), adapt and extend the classic Bron-Kerbosch algorithm [BK73] to enumerate them, prove its correctness (Proposition 7.6), and present a worst-case running time analysis of our new algorithm (Proposition 7.12). We further introduce a measure for sparsity for temporal graphs called Δ-*slice degeneracy* (Definition 7.7), which is a temporal adaptation of the degeneracy for static graphs [LW70]. Our running time analysis shows that our algorithm has an FPT running time if s is constant and when the problem is parameterized by the Δ-slice degeneracy of the input temporal graph (Theorem 7.3).

The chapter is organized as follows. In Section 7.2 we introduce some additional definitions and notations. We formally define Δ-cliques and Δ-s-plexes and define decision versions of our problem. We discuss the adaptation of degeneracy to temporal graphs and give the definition of Δ-slice degeneracy. We further give some easy to observe intractability results and, lastly, we briefly explain the original Bron-Kerbosch algorithm which serves as a role model for our algorithm. In Section 7.3, we present our adaptation of the Bron-Kerbosch algorithm for enumerating all maximal Δ-s-plexes in temporal graphs. After a detailed description of the algorithm, we

prove its correctness and analyze its running time. We further utilize the parameter "Δ-slice degeneracy of a temporal graph" to upper-bound the number of maximal Δ-s-plexes of a temporal graph and to show the fixed-parameter tractability of our problem. We conclude in Section 7.4.

7.1.3 Further Contributions of the Papers this Chapter is Based on

Himmel et al. [Him+17] focus on the problem of enumerating Δ-cliques. They propose several pivoting techniques to reduce the number of recursive calls of their algorithm. Furthermore, they give an implementation of the algorithm and an experimental evaluation showing that their approach works well in practice.

Bentert et al. [Ben+19] further give an implementation of the algorithm to enumerate Δ-s-plexes and propose additional heuristic speed-ups. They also present an experimental evaluation of their algorithm showing that in practice, enumerating Δ-s-plexes is only possible for small values of s.

7.2 Preliminaries

In this section, we provide additional notation for intervals and sets of intervals. We introduce further concepts related to temporal graphs and give the formal problem definitions of Temporal Clique and Temporal s-Plex and some preliminary computational hardness results. We also give a short description of the classic Bron-Kerbosch algorithm [BK73].

7.2.1 Intervals and Sets of Intervals

We introduce various concepts related to intervals of natural numbers that we need in this chapter. The most central one is that of a *set $\mathfrak{I}$ of intervals*. On a mathematical level, a set $\mathfrak{I}$ of intervals is a finite subset of the natural numbers. Additionally, we assume that in the algorithm we present, sets of intervals are always stored as ordered lists. The name "set of intervals" stems from the fact that we interpret these sets of intervals as literal sets of intervals, even though from a mathematical standpoint, they are just finite sets of integers. According to this intuition, we define the notion of an interval being an element of a set of intervals as follows.

Definition 7.1. An interval $I = [a, b]$ is *contained* in a set of intervals $\mathfrak{I} \subset \mathbb{N}$ if $I \subseteq \mathfrak{I}$ and there is no larger interval $I' \supset I$ with $I' \subseteq \mathfrak{I}$. If I is contained in $\mathfrak{I}$, then we denote this with $I \sqsubseteq \mathfrak{I}$.

We refer to a tuple (v, I) with v being a vertex and I being an interval as a *vertex-interval pair* and we refer to a tuple $(v, \mathfrak{I})$ with a vertex v and a set $\mathfrak{I}$ of intervals as a

vertex-interval-set pair. For a set A of vertex-interval-set pairs, we define $V(A)$ to be the set of all vertices of vertex-interval-set pairs in A. We further define the following notation.

Definition 7.2. Let X, Y be sets of vertex-interval-set pairs, let (v, I) be a vertex-interval pair, let $(v, \mathfrak{I}_v)$ be a vertex-interval-set pair, let $\mathfrak{I}$ be a set of intervals, and let $V' \subseteq V$ be a set of vertices:

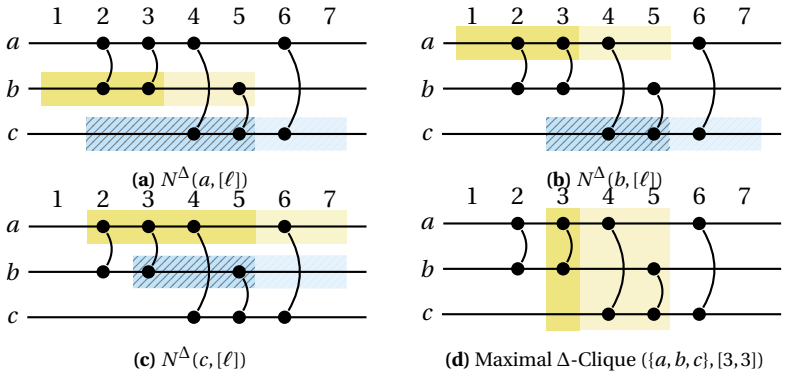
- $X[\mathfrak{I}] := \{(u, \mathfrak{I}_u \cap \mathfrak{I}) \mid (u, \mathfrak{I}_u) \in X\}$;
- $X[V'] := \{(u, \mathfrak{I}) \mid (u, \mathfrak{I}) \in X \wedge u \in V'\}$;
- $(v, \mathfrak{I}_v) \sqcup X := \begin{cases} X \cup \{(v, \mathfrak{I}_v)\} & \text{if } \neg\exists (v, \mathfrak{I}_v^X) \in X, \\ (X \setminus \{(v, \mathfrak{I}_v^X)\}) \cup \{(v, \mathfrak{I}_v \cup \mathfrak{I}_v^X)\} & \text{otherwise;} \end{cases}$
- $X \sqcap Y := \{(u, \mathfrak{I}_u^X \cap \mathfrak{I}_u^Y) \mid (u, \mathfrak{I}_u^X) \in X \wedge (u, \mathfrak{I}_u^Y) \in Y\}$;
- $(v, \mathfrak{I}_v) \sqcap X := \{(v, \mathfrak{I}_v)\} \sqcap X$; and
- $(v, I) \sqsubseteq X := (v, \{I\}) \in (v, \{I\}) \sqcap X$.

7.2.2 Δ-Neighborhoods and Δ-Non-Neighborhoods

In order to properly define Δ-cliques and Δ-s-plexes, we need to adjust the notion of vertex neighborhoods and non-neighborhoods from static to temporal graphs. Instead of just considering the incident edges of a vertex at one time step, we consider all incident edges within a Δ-window. We say that two vertices v and w are *neighbors* in the Δ-window W_i^Δ if there is an edge $\{v, w\} \in E_t$ with $t \in W_i^\Delta$. The Δ-*neighborhood* $N^\Delta(v, i)$ of a vertex $v \in V$ and a Δ-window W_i^Δ is the set of all neighbors of v in W_i^Δ. Accordingly, we define the Δ-*neighborhood* $N^\Delta(v, \mathfrak{I}_v)$ of a vertex-interval-set pair $(v, \mathfrak{I}_v)$ as the set of vertex-interval-set pairs $(w, \mathfrak{I}_w)$ with maximal $\mathfrak{I}_w$ such that $\mathfrak{I}_w \subseteq \mathfrak{I}_v$ and there is an edge between v and w in every Δ-window W_i^Δ with $i \in \mathfrak{I}_w$. See Figures 7.1a to 7.1c for an example. More formally, we arrive at the following definition.

Definition 7.3 (Δ-Neighborhood)**.** Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a temporal graph, let $v \in V$ be a vertex, and let W_i^Δ, $i \in [\ell - \Delta + 1]$, be a Δ-window of $\mathcal{G}$. The Δ-*neighborhood* of v in W_i^Δ is defined as

$$N^\Delta(v, i) := \{w \in V \mid \exists t \in W_i^\Delta : \{v, w\} \in E_t\}.$$



(a) $N^{\Delta}(a,[\ell])$ **(b)** $N^{\Delta}(b,[\ell])$

(c) $N^{\Delta}(c,[\ell])$ **(d)** Maximal Δ-Clique $(\{a,b,c\},[3,3])$

Figure 7.1: Δ-Neighborhoods and a Δ-clique of a temporal graph with $\Delta = 3$. The lifetime of the temporal graph is $\ell = 7$. The elements of the Δ-neighborhoods of vertices a, b, and c are visualized in Figures 7.1a to 7.1c, respectively, in yellow and blue (hatched). Light shaded areas correspond to the time steps in the respective last Δ-window. Figure 7.1d shows a maximal Δ-clique shaded in yellow, where the light yellow part corresponds to the time steps in the last Δ-window.

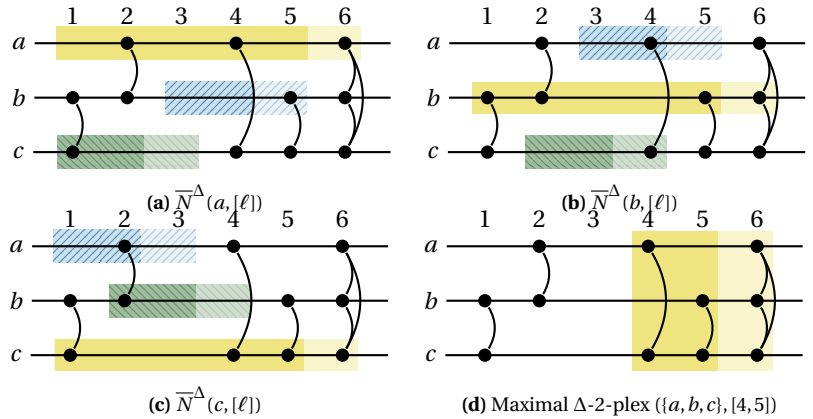
Let $(v,\mathfrak{I}_v)$ be a vertex-interval-set pair of $\mathcal{G}$. The Δ-*neighborhood* of v in $\mathfrak{I}_v$ is defined as

$$N^{\Delta}(v,\mathfrak{I}_v) := \{(w,\mathfrak{I}_w) \mid w \in V \wedge \mathfrak{I}_w = \bigcup_{i \in \mathfrak{I}_v} \{i \mid w \in N^{\Delta}(v,i)\}\}.$$

Notice that being a Δ-neighbor of another vertex is a symmetric relation. If $(w,I) \sqsubseteq N^{\Delta}(v,\mathfrak{I})$, then we say that w is a Δ-*neighbor* of v during the time interval I.

Accordingly, we say that v and w are *non-neighbors* in W_i^{Δ} if there exists no edge $\{v,w\} \in E_t$ with $t \in W_i^{\Delta}$. The Δ-*non-neighborhood* $\overline{N}^{\Delta}(v,i)$ of a vertex $v \in V$ and a Δ-window W_i^{Δ} is the set of all non-neighbors of v in W_i^{Δ}. Accordingly, we define the Δ-*non-neighborhood* $\overline{N}^{\Delta}(v,\mathfrak{I}_v)$ of a vertex-interval-set pair $(v,\mathfrak{I}_v)$ as the set of vertex-interval-set pairs $(w,\mathfrak{I}_w)$ with maximal $\mathfrak{I}_w$ such that $\mathfrak{I}_w \subseteq \mathfrak{I}_v$ and there is no edge between v and w in any Δ-window W_i^{Δ} with $i \in \mathfrak{I}_w$. See Figures 7.2a to 7.2c for an example. More formally, we arrive at the following definition.

Definition 7.4 (Δ-Non-Neighborhood)**.** Let $\mathcal{G} = (V,(E_i)_{i\in[\ell]})$ be a temporal graph, let $v \in V$ be a vertex, and let W_i^{Δ}, $i \in [\ell - \Delta + 1]$, be a Δ-window of $\mathcal{G}$. The Δ-*non-*



(a) $\overline{N}^{\Delta}(a,[\ell])$ **(b)** $\overline{N}^{\Delta}(b,[\ell])$

(c) $\overline{N}^{\Delta}(c,[\ell])$ **(d)** Maximal Δ-2-plex $(\{a,b,c\},[4,5])$

Figure 7.2: Δ-Non-neighborhoods and a Δ-2-plex of a temporal graph with $\Delta = 2$. The lifetime of the temporal graph is $\ell = 6$. The elements of the Δ-non-neighborhoods of vertices a, b, and c are visualized in Figures 7.1a to 7.1c, respectively, in yellow, blue (hatched), and green (hatched). Light shaded areas correspond to the time steps in the respective last Δ-window. In Figure 7.2a, we can see for example that b within time interval $[3,4]$ is in the Δ-non-neighborhood of a because in the Δ-windows $W_3^{\Delta}([3,4])$ and $W_4^{\Delta}([4,5])$ there are no time edges between a and b. Figure 7.2d shows a maximal Δ-2-plex shaded in yellow, where the light yellow part corresponds to the time steps in the last Δ-window. The tuple $(\{a,b,c\},[4,5])$ is a Δ-2-plex because in Δ-windows $W_4^{\Delta}([4,5])$ and $W_5^{\Delta}([5,6])$ each vertex has at most two non-neighbors (including itself).

neighborhood of v in W_i^{Δ} is defined as

$$\overline{N}^{\Delta}(v,i) := \{w \in V \mid \forall\, t \in W_i^{\Delta} : \{v,w\} \notin E_t\}.$$

Let $(v,\mathfrak{I}_v)$ be a vertex-interval-set pair of $\mathcal{G}$. The Δ-*non-neighborhood* of v in $\mathfrak{I}_v$ is defined as

$$\overline{N}^{\Delta}(v,\mathfrak{I}_v) := \{(w,\mathfrak{I}_w) \mid w \in V \wedge \mathfrak{I}_w = \bigcup_{i \in \mathfrak{I}_v} \{i \mid w \in \overline{N}^{\Delta}(v,i)\}\}.$$

7.2.3 Δ-Cliques and Δ-s-Plexes

A straightforward adaptation of a clique to the temporal setting is to additionally assign a lifetime $I = [a,b]$ to it, that is, the largest time interval such that the clique exists in each time step in said interval. However, this model is often too restrictive

for real-world data. For example, if the subject matter of examination is e-mail traffic and the data set includes e-mails with time stamps including seconds, we are not interested in people who sent e-mails to each other every second over a certain time interval, but we would like to know which groups of people were in contact with each other, say, at least every seven days over months. One possible approach would be to generalize the time stamps, taking into account only the week an e-mail was sent, resulting in a loss of accuracy in the data set. The constraint of each pair of vertices being connected in each time step can be relaxed by introducing an additional parameter Δ, quantifying how many time steps may be skipped between two connections of any vertex pair. These so-called Δ-cliques were introduced by Viard, Latapy, and Magnien [VLM16] and are formally defined as follows.

Definition 7.5 (Δ-Clique)**.** Given a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, an integer $\Delta \in \mathbb{N}$, a subset $C \subseteq V$ of vertices, and an interval $I = [a, b] \subseteq [\ell - \Delta + 1]$, then $R = (C, I)$ is a Δ-*clique* if for all $v, w \in C$ and all $i \in I$ it holds that $w \in N^{\Delta}(v, i)$.

In other words, for a Δ-clique $R = (C, I)$ all pairs of vertices in C interact with each other at least once in every Δ-window that starts at a time step in the time interval I. See Figure 7.1 for an illustration of a Δ-clique. It is evident that the parameter Δ is a measurement of the intensity of interactions in Δ-cliques. Small Δ-values imply that the interaction between vertices in a Δ-clique has to be more frequent than in the case of large Δ-values. The choice of Δ depends on the data set and the purpose of the analysis.

We can also consider Δ-cliques from another point of view. For a given temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, an integer $\Delta \in \mathbb{N}$, and a time step $t \in [\ell - \Delta + 1]$, the static graph $G_t^{\Delta} = (V, E_{W_t^{\Delta}})$ describes all contacts that appear within the Δ-window W_t^{Δ} in the temporal graph $\mathcal{G}$. The existence of a Δ-clique $R = (C, I = [a, b])$ implies that all vertices in C form a clique in all static graphs G_t^{Δ} with $t \in I$. This implies that all vertices in C are pairwise connected to each other in the static graphs of all sliding, Δ-sized time windows from time a until b.

We can now formally define the (decision) problem of finding a Δ-clique with a certain minimum amount of vertices and a certain minimum lifetime in a given temporal graph $\mathcal{G}$.

Temporal Clique	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and three integers $k, t \in \mathbb{N}$, and $\Delta \leq \ell$.
Question:	Does $\mathcal{G}$ contain a Δ-clique $(C, [a, b])$ with $\lvert C\rvert \geq k$ and $b - a \geq t$?

TEMPORAL CLIQUE clearly generalizes the classic NP-complete CLIQUE problem [Kar72]. It is further easy to check that, given a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, it can be verified in polynomial time whether $R = (C, I)$ with $C \subseteq V$ and $I \subseteq [\ell - \Delta + 1]$ is a Δ-clique in $\mathcal{G}$. Hence, we have that TEMPORAL CLIQUE is NP-complete for each fixed $t \geq 1$, $\Delta \geq 1$, and $\ell \geq \Delta$.

In an enumeration setting, we are most interested in Δ-cliques that are not "contained" in any other Δ-clique. For this we also need to adapt the notion of maximality to the temporal setting [VLM16]. Let $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ be a temporal graph. We call a Δ-clique $R = (C, I)$ in $\mathcal{G}$ *vertex-maximal* if there is no vertex $v \in V \setminus C$ such that $(C \cup \{v\}, I)$ is a Δ-clique in $\mathcal{G}$. Intuitively, a Δ-clique is vertex-maximal if we cannot add any vertex to C without having to decrease the clique's lifetime I. We say that a Δ-clique $R = (C, I)$ is *time-maximal* if there is no $I \subset I' \subseteq [\ell - \Delta + 1]$ such that $R' = (C, I')$ is a Δ-clique in $\mathcal{G}$. Intuitively, a Δ-clique is time-maximal if we cannot increase the lifetime I without having to remove vertices from C. We call a Δ-clique *maximal* if it is both vertex-maximal and time-maximal.

We define a Δ-s-*plex* as a straightforward relaxation of a Δ-clique. A Δ-s-plex consists of a set C of vertices and a lifetime $I = [a, b]$. Analogously to s-plexes in static graphs [SF78], Δ-s-plexes are defined so that each vertex in the vertex set C of the Δ-s-plex must have at least $|C| - s$ neighbors in each Δ-window W_i^Δ with $i \in I$. In other words, for every vertex $v \in C$ there are at most s vertices both in C and in the Δ-non-neighborhood of (v, i) for each $i \in I$. Note that a Δ-1-plex is a Δ-clique. See Figure 7.2d for an illustration of a Δ-s-plex. The formal definition is as follows.

Definition 7.6 (Δ-s-Plex)**.** Given a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, an integer $\Delta \in \mathbb{N}$, a subset $C \subseteq V$ of vertices, and an interval $I = [a, b] \subseteq [\ell - \Delta + 1]$, then $R = (C, I)$ is a Δ-s-*plex* if for all $v \in C$ and all $i \in I$ it holds that $|\overline{N}^\Delta(v, i) \cap C| \leq s$.

We can now formally define the (decision) problem of finding a Δ-s-plex with a certain minimum amount of vertices and a certain minimum lifetime in a given temporal graph $\mathcal{G}$.

TEMPORAL s-PLEX	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and three integers $k, t \in \mathbb{N}$, and $\Delta \leq \ell$.
Question:	Does $\mathcal{G}$ contain a Δ-s-plex $(C, [a, b])$ with $\lvert C\rvert \geq k$ and $b - a \geq t$?

By definition we have that TEMPORAL s-PLEX generalizes the NP-complete s-PLEX problem [BBH11]. It is further easy to check that, given a temporal graph $\mathcal{G} =$

$(V,(E_i)_{i\in[\ell]})$, it can be verified in polynomial time whether $R=(C,I)$ with $C\subseteq V$ and $I\subseteq[\ell-\Delta+1]$ is a Δ-s-plex in $\mathcal{G}$. Hence, we have that Temporal s-Plex is NP-complete for each fixed $s\geq 1$, $t\geq 1$, $\Delta\geq 1$, and $\ell\geq\Delta$.

In the enumeration setting, we focus on finding maximal Δ-s-plexes. Analogously to the case of Δ-cliques, there is both vertex-maximality and time-maximality. Given a temporal graph $\mathcal{G}=(V,(E_i)_{i\in[\ell]})$, a Δ-s-plex $R=(C,I)$ is *vertex-maximal* if and only if there is no vertex $v\in V\setminus C$ such that $(C\cup\{v\},I)$ is a Δ-s-plex in $\mathcal{G}$. Intuitively, a Δ-s-plex is vertex-maximal if no other vertex can be added to it without decreasing its lifetime. We say that a Δ-s-plex $R=(C,I)$ is *time-maximal* if and only if there is no $I\subset I'\subseteq[\ell-\Delta+1]$ such that $R'=(C,I')$ is a Δ-s-plex in $\mathcal{G}$. Intuitively, a Δ-s-plex is time-maximal if we cannot increase its lifetime without removing a vertex from it. We call a Δ-s-plex *maximal* if it is both vertex-maximal and time-maximal.

7.2.4 Degeneracy of Temporal Graphs

Degeneracy is a measure of sparsity for static graphs [LW70] (for a definition see Section 2.4). Real-world instances of static graphs (especially social networks) are often sparse, which results in a small degeneracy value [ELS13]. This motivates using the degeneracy of the input graph as a parameter for graph problems that are motivated in the context of for example social network analysis, such as Clique.

It is easy to see that the maximum clique size of a graph with degeneracy d is at most $d+1$: If there is a clique of size at least $d+2$, then the vertices of this clique would form an induced subgraph in which every vertex v of the clique has a degree larger than d. Indeed, it is known that Clique parameterized by the degeneracy of the input graph is fixed-parameter tractable [ELS13].

This motivates a parameterized complexity analysis of Temporal Clique parameterized by a temporal analogue of degeneracy. There are several canonical ways to transfer degeneracy from the static to the temporal setting. The two most obvious are, given a temporal graph,

1. to consider the maximum of the degeneracy values of all layers, or
2. to consider the degeneracy value of the underlying graph.

It is clear that the former is smaller than the latter. However, for our problem, we define a temporal adaptation of degeneracy that lies between the two options above, the so-called Δ-slice degeneracy. Intuitively, it measures the maximum degeneracy value of the union graphs of all Δ-windows of a temporal graph.

Definition 7.7 (Δ-Slice Degeneracy)**.** A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ has Δ-*slice degeneracy* d if for all $t \in [\ell - \Delta + 1]$ it holds that $G_{W_t^\Delta} = (V, E_{W_t^\Delta})$ has degeneracy at most d.

We remark that, from a parameterized point of view, the Δ-slice degeneracy is at most as large as the maximum degeneracy of all layers combined with Δ. This follows from the observation that the degeneracy can be lower-bounded by the arboricity[15] and upper-bounded by twice the arboricity [Nas61, Nas64, SW19, Tut61]. Further, we have that it is obvious that the arboricity of the union graph of a Δ-window $G_{W_t^\Delta} = (V, E_{W_t^\Delta})$ is at most as large as the sum of the arboricities of the layers in that Δ-window. It can, however, be much smaller.

7.2.5 Hardness Results

It this section, we give some easy to observe hardness results that show that our main fixed-parameter tractability result (Theorem 7.3) presumably cannot be significantly improved from a classification standpoint.

Observation 7.1. Temporal Clique *and* Temporal s-Plex *for all* $s \geq 1$ *are NP-complete even if the degeneracy of every layer of the input temporal graph is one.*

Proof. This can be shown by an easy polynomial-time reduction from the NP-complete problems Clique [Kar72] and s-Plex [BBH11], respectively. Given an input instance $(H = (U, F), k)$ of Clique or s-Plex, respectively, we construct a temporal graph $\mathcal{G}$ with U as vertex set and where for each edge $e \in F$ we create one layer that only contains e. It follows that $\ell = |F|$. We set $\Delta = \ell$ and $t = 1$. It is easy to see that $(\mathcal{G}, k, t, \Delta)$ is a YES-instance of the respective temporal problem if and only if (H, k) is a YES-instance of the respective original problem. □

This observation implies that we cannot hope to use the maximum degeneracy value of any layer as a parameter to show fixed-parameter tractability unless P = NP.

We can further show that we presumably cannot hope to improve our main fixed-parameter tractability result to a polynomial kernel, even if we increase our parameter to the number $|V|$ of vertices.

Proposition 7.2. Temporal Clique *and* Temporal s-Plex *for all* $s \geq 1$ *parameterized by the number* $|V|$ *of vertices do not admit a polynomial kernel for all* $\Delta \geq 1$ *and* $t \geq 1$ *unless NP* ⊆ *coNP/poly.*

[15] The *arboricity* of an undirected graph G is the minimum number of forests into which its edges can be partitioned.

Proof. We provide an OR-cross-composition (for a definition see Section 2.3) from the NP-complete problems CLIQUE [Kar72] and s-PLEX [BBH11] to TEMPORAL CLIQUE and TEMPORAL s-PLEX, respectively. Intuitively, we can just string together instances in the time axis such that the large instance contains a Δ-clique or Δ-s-plex if and only if one of the original instances is a YES-instance.

We define an equivalence relation R as follows: Two instances $(H = (U, F), k)$ and $(H' = (U', F'), k')$ are equivalent under R if and only if $|U| = |U'|$ and $k = k'$. Clearly, R is a polynomial equivalence relation.

Now let $(H_1 = (U_1, F_1), k_1), \ldots, (H_n = (U_n, F_n), k_n)$ be R-equivalent instances of CLIQUE or s-PLEX, respectively. We arbitrarily identify the vertices of the instances, that is, let $V = U_1 = \ldots = U_n$ and create a temporal graph $\mathcal{G} = (V, F_1, F_2, \ldots, F_n)$. Further, we set $\Delta = 1$, $t = 1$ and $k = k_1$.

This instance can be constructed in polynomial time and the number of vertices is the same as the vertices of the input instances, hence $|V|$ is polynomially upper-bounded by the maximum size of an input instance. Furthermore, it is easy to check that $\mathcal{G}$ contains a Δ-clique or Δ-s-plex with at least k vertices and a lifetime of one if and only if there is an $i \in [n]$ such that H_i contains a clique or an s-plex with k vertices, respectively. The result follows. □

7.2.6 The Classic Bron-Kerbosch Algorithm

The Bron-Kerbosch algorithm is a classic algorithm that enumerates all maximal cliques in a static graph and hence also solves the CLIQUE problem [BK73]. It is a simple yet clever backtracking algorithm, which can be made to perform very well on real-world networks [ELS13]. We give a short description here since our adaptation inherits several ideas of this algorithm.

See Algorithm 1 for pseudo-code of the Bron-Kerbosch algorithm. The algorithm maintains three distinct sets of vertices. The first set R contains the current clique. The other two sets P and X contain the vertices that can be added to R such that R is still a clique. The set P contains all candidate vertices which have not been considered in previous iterations, while the set X contains vertices that have been considered before. In each recursive call, the algorithm first checks whether the current clique R is maximal, that is, whether $P \cup X = \emptyset$. If so, then it adds R to the solution, otherwise it iterates through all vertices $v \in P$, adds v to R, and recursively calls itself with updated sets P' and X' where all vertices that are not adjacent to v are removed. Afterwards, it removes v from P and adds it to X. The initial call is with $P = V$ and $R = X = \emptyset$.

Algorithm 1 Enumerating all Maximal Cliques

```
1: function BronKerbosch(P, R, X)
   ▷ R: a clique.
   ▷ P ∪ X: set of all vertices v such that R ∪ {v} is a clique and where
     • vertices in P have not yet been considered as additions to R and
     • vertices in X already have been considered in earlier iterations.
2:     if P ∪ X = ∅ then
3:         add R to the solution
4:     end if
5:     for v ∈ P do
6:         BronKerbosch(P ∩ N(v), R ∪ {v}, X ∩ N(v)))
7:         P ← P \ {v}
8:         X ← X ∪ {v}
9:     end for
10: end function
```

7.3 Enumerating Temporal Cliques and s-Plexes

In this section we give an algorithm that shows that Temporal Clique and Temporal s-Plex with fixed s are fixed-parameter tractable when parameterized with the Δ-slice degeneracy of the input temporal graph. Formally, we prove the following.

Theorem 7.3. Temporal s-Plex *can be solved in* $O(\binom{|V|}{s-1} \cdot 2^{d+s} \cdot \min\{(\sum_{i=1}^{\ell} |E_i|)^2, \ell^2\} \cdot |V|^3)$ *time, where* d *is the* Δ*-slice degeneracy of the input graph.*

The algorithm we present even enumerates all maximal Δ-cliques or Δ-s-plexes. Note that since Temporal Clique is equivalent to Temporal 1-Plex, we focus on the more general Temporal s-Plex problem in the remainder of this chapter.

The classic Bron-Kerbosch algorithm enumerates all maximal cliques in a static graph [BK73] and has been adapted to enumerate s-plexes in static graphs [Con+18, WP07]. In the following, we describe how to adapt the Bron-Kerbosch algorithm to enumerate maximal Δ-s-plexes in temporal graphs. We call the new algorithm Δ-s-BronKerbosch, see Algorithms 2 to 4 for pseudocodes.

The Bron-Kerbosch algorithm is a recursive backtracking algorithm that works on the idea of maintaining a current clique and two candidate sets containing vertices that may be added to the clique, one that contains vertices that are recursively added to the clique and one that contains vertices that already have been part of the clique in an earlier recursive call. The latter prevents the algorithm from discovering

Algorithm 2 Enumerating all Maximal Δ-s-Plexes

function Δ-s-BronKerbosch($P, R = (C, \mathfrak{I}), X, B$)
▷ $R = (C, \mathfrak{I})$: for every $I \sqsubseteq \mathfrak{I}$, (C, I) is a time-maximal Δ-s-plex.
▷ $P \cup X$: set of all vertex-interval-set pairs $(v, \mathfrak{I}_v)$ such that for all $I_v \sqsubseteq \mathfrak{I}_v$ it holds that $I_v \subseteq \mathfrak{I}$ and $(C \cup \{v\}, I_v)$ is a time-maximal Δ-s-plex and where
- vertex-interval-set pairs in P have not yet been considered as additions to R and
- vertex-interval-set pairs in X have been considered in earlier iterations.

▷ $B\colon V \times [\ell - \Delta + 1] \mapsto \mathbb{N}$ with $B(v,t) = |\overline{N}^{\Delta}(v,t) \cap C|$, that is, function B returns for every vertex $v \in V$ and every time step $t \in [\ell - \Delta + 1]$ the number of non-neighbors of v from C in Δ-window W_t^{Δ}.
  for $I \sqsubseteq \mathfrak{I}$ **do**
    if $\forall (v, \mathfrak{I}_v) \in P \cup X$ and $\forall I_v \sqsubseteq \mathfrak{I}_v\colon I_v \neq I$ **then**
      add (C, I) to the solution
    end if
  end for
  for $(v, \mathfrak{I}_v) \in P$ **do**
    $C' \leftarrow C \cup \{v\}$
    $\mathfrak{I}' \leftarrow \mathfrak{I}_v$
  ▷ Update of the function B and the sets P and X for the new set of Δ-s-plexes $(C', \mathfrak{I}')$. Crit contains all pairs $(v, \mathfrak{I}_v)$ where v has exactly s non-neighbors from C' in $I \sqsubseteq \mathfrak{I}_v$.
    $B', \text{Crit} \leftarrow$ UpdatePool$(B, C', (v, \mathfrak{I}_v))$
    $P' \leftarrow$ Update$(P, C', \text{Crit}, (v, \mathfrak{I}_v))$
    $X' \leftarrow$ Update$(X, C', \text{Crit}, (v, \mathfrak{I}_v))$
    Δ-s-BronKerbosch$(P', R' = (C', \mathfrak{I}'), X', B')$
    $P \leftarrow P \setminus \{(v, \mathfrak{I}_v)\}$
    $X \leftarrow X \cup \{(v, \mathfrak{I}_v)\}$
  end for
end function

a clique several times. If those sets are empty, we know that the clique must be maximal. In the case of s-plexes one need to introduce additional data structures that essentially count the number of "missing neighbors" for every vertex in the current s-plex. This basic idea stays the same in the temporal case, with the main difference that apart from vertices, we also have to manage time intervals. In the following, we give a more formal explanation on how our temporal adaptation of the Bron-Kerbosch algorithm works.

The input of Δ-s-BronKerbosch consists of two sets P and X of vertex-interval-set pairs, an implicit set of current time-maximal Δ-s-plexes $R = (C, \mathfrak{I})$, as well as a *pool* B,

Algorithm 3 Updating the Pool Function B

function UPDATEPOOL($B, C, (v, \mathfrak{I}_v)$)
▷ B: $V \times [\ell - \Delta + 1] \to \mathbb{N}$ with $B(v,t) = |\overline{N}^{\Delta}(v,t) \cap C|$.
▷ $C, (v, \mathfrak{I}_v)$: $v \in C$ and for every $I \sqsubseteq \mathfrak{I}_v$ it holds that (C, I) is a time-maximal Δ-s-plex.
▷ Update function B for $v \in C$; store critical vertex-interval-set pairs.
  Crit $\leftarrow \{(c, \emptyset) \mid c \in C \cup V(P)\}$
  $B' \leftarrow B$
  for $(w, \mathfrak{I}_w) \in P \cup X \cup \{(c, \mathfrak{I}_v) \mid c \in C\}$ **do**
  ▷ Function B only changes if a vertex is in the Δ-non-neighborhood of v during $\mathfrak{I}_v$.
    $(w, \mathfrak{I}_{\text{crit}}) \leftarrow (w, \mathfrak{I}_w) \sqcap \overline{N}^{\Delta}(v, \mathfrak{I}_v)$
    for $t \in \mathfrak{I}_{\text{crit}}$ **do**
      $B'(w,t) \leftarrow B'(w,t) + 1$
    ▷ If a vertex v already has s non-neighbors from C in Δ-window W_t^{Δ}, then this vertex is critical.
      if $B'(w,t) = s$ **then**
        Crit $\leftarrow (w, \{t\}) \sqcup$ Crit
      end if
    end for
  end for
  return B', Crit
end function

which is a data structure that keeps track of the missing neighbors of each vertex of the current Δ-s-plexes. Herein, C is the set of vertices of the Δ-s-plexes and $\mathfrak{I}$ is a set of intervals such that for all $I \sqsubseteq \mathfrak{I}$ we have that (C, I) forms a time-maximal Δ-s-plex. The pool is an auxiliary data structure that stores the number of Δ-non-neighbors of the vertices of the Δ-s-plex in any Δ-window. While in the original Bron-Kerbosch algorithm the sets P and X contain the common neighborhood of all vertices in R, our sets P and X contain all vertices v with interval sets $\mathfrak{I}_v$ such that for all $I_v \sqsubseteq \mathfrak{I}_v$ it holds that $I_v \subseteq \mathfrak{I}$ and $(C \cup \{v\}, I_v)$ is a time-maximal Δ-s-plex. These vertices cannot be contained in the Δ-non-neighborhood of more than $s-1$ other vertices of C in each Δ-window W_i^{Δ} with $i \in I_v$. They can neither be contained in the Δ-non-neighborhood of a vertex $w \in C$ during its *critical* intervals, that is, the intervals where w has exactly s Δ-non-neighbors in C (including w itself). To maintain these properties after expanding the current Δ-s-plex, we update the pool B with the UPDATEPOOL procedure (Algorithm 3) after adding a new vertex $v \in V(P)$ to C and then update the sets P and X with the UPDATE procedure (Algorithm 4). For each vertex in

Algorithm 4 Updating all Vertex-Interval-Sets

function UPDATE($P, C, \text{Crit}, (v, \mathfrak{I}_v)$)
▷ Update P such that for all $(w, \mathfrak{I}_w)$ and all $I_w \sqsubset \mathfrak{I}_w$ with $I_w \subseteq \mathfrak{I}_v$ it holds that $(C \cup \{w\}, I_w)$ is a time-maximal Δ-s-plex.
  $P_{\text{reduced}} \leftarrow P[\mathfrak{I}_v]$
  $P' \leftarrow \emptyset$
  for $(w, \mathfrak{I}_w) \in P_{\text{reduced}}[V(P) \setminus \{v\}]$ **do**
  ▷ If w has a non-neighbor $u \in C$ in some W_i^Δ with $i \in \mathfrak{I}_w$ and u is critical in W_i^Δ, then we cannot add w to C at time step i.
    for $(u, \mathfrak{I}_u) \in \text{Crit}[C \cup \{w\}] \sqcap \overline{N}^\Delta(w, \mathfrak{I}_w)$ **do**
      $\mathfrak{I}_w \leftarrow \mathfrak{I}_w \setminus \mathfrak{I}_u$
    end for
    $P' \leftarrow P' \cup \{(w, \mathfrak{I}_w)\}$
  end for
  return P'
end function

$V(P) \cup V(X) \cup C$, we save the number of Δ-non-neighbors of vertices in C for each Δ-window in the pool B. We iterate through all vertex-interval-set pairs $(v, \mathfrak{I}_v) \in P$, call the UPDATEPOOL and UPDATE procedures, and then do a recursive call with the updated sets R', P', and X'.

For a given temporal graph $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$, the input for the initial call of Δ-s-BRONKERBOSCH to enumerate all maximal Δ-s-plexes is $P = \{(v, [\ell - \Delta + 1]) \mid v \in V\}$, $X = \emptyset$, $R = (\emptyset, [\ell - \Delta + 1])$, and $B(v, i) = 0$ for all $v \in V$ and $i \in [\ell - \Delta + 1]$. In the remainder of this chapter, we always assume this initial call of the algorithm, that is, the correctness of all theorem, proposition, and lemma statements is conditioned on this assumption.

7.3.1 Correctness of Δ-s-BRONKERBOSCH

In this section we prove the correctness of our proposed algorithm Δ-s-BRONKERBOSCH (Algorithm 2).

We start by showing that the pools are correctly maintained by the UPDATEPOOL function (see Algorithm 3), that is, the value of each pool on each relevant Δ-window is equal to the amount of non-neighbors in the current Δ-s-plex R. Afterwards we show an invariance for all recursive calls of Δ-s-BRONKERBOSCH that then allows us to prove the correctness of Δ-s-BRONKERBOSCH. Formally, we show the following property of the recursive calls of Δ-s-BRONKERBOSCH.

Lemma 7.4. *In each recursive call of* Δ-s-BronKerbosch$(P, R = (C, \mathfrak{I}), X, B)$*, for each vertex-interval-set-pair* $(w, \mathfrak{I}_w) \in P \cup X \cup \{(c, \mathfrak{I}) \mid c \in C\}$ *and each* $t \in \mathfrak{I}_w$ *we have*

$$B(w,t) = |\overline{N}^{\Delta}(w,t) \cap C|.$$

Proof. We prove this lemma by induction on the recursion depth, that is, the number $|C|$ of vertices of the Δ-s-plex in the current recursive call.

Initially, the Δ-s-BronKerbosch (Algorithm 2) is called with $R = (\emptyset, [\ell - \Delta + 1])$ and the pool B is initialized for all $w \in V$ and $t \in [\ell - \Delta + 1]$ with $B(w,t) = 0 = |\overline{N}^{\Delta}(w,t) \cap \emptyset|$. Hence, the initialization is correct.

Now let us assume that for a recursive call Δ-s-BronKerbosch$(P, R = (C, \mathfrak{I}), X, B)$ it holds that for each vertex-interval-set-pair $(v, \mathfrak{I}_v) \in P \cup X \cup \{(c, \mathfrak{I}) \mid c \in C\}$ and $t \in \mathfrak{I}_v$ we have $B(v,t) = |\overline{N}^{\Delta}((,v),t) \cap C|$. Let $(v, \mathfrak{I}_v) \in P$ be a vertex-interval-set-pair added to R (Lines 8 and 9 of Algorithm 2), that is, $R' = (C \cup \{v\}, \mathfrak{I}_v)$. Now for each $(w, \mathfrak{I}_w) \in P \cup X \cup \{(c, \mathfrak{I}_v) \mid c \in C\}$ and $t \in \mathfrak{I}_w \cap \mathfrak{I}_v$ the pool-function value $B(w,t)$ is increased by one if w and v are non-neighbors in the Δ-window W_t^{Δ} in Lines 4 to 7 in the UpdatePool procedure (Algorithm 3), that is,

$$\begin{aligned} B'(w,t) &= \begin{cases} B(w,t)+1 & \text{if } v \in \overline{N}^{\Delta}(w,t) \\ B(w,t) & \text{else} \end{cases} \\ &= B(w,t) + |\overline{N}^{\Delta}(w,t) \cap \{v\}| \\ &= |\overline{N}^{\Delta}(w,t) \cap C| + |\overline{N}^{\Delta}(w,t) \cap \{v\}| \\ &= |\overline{N}^{\Delta}(w,t) \cap (C \cup \{v\})|. \end{aligned}$$

The last equality holds because $v \notin C$. Next in the Update procedure (Algorithm 4), the sets P and X are updated according to the new Δ-s-plexes in R'. For each vertex-interval-set-pair $(w, \mathfrak{I}'_w)$ in the new sets P' and X' it holds that $(w, \mathfrak{I}_w) \in P \cup X$ with $\mathfrak{I}'_w \subseteq \mathfrak{I}_w$, $\mathfrak{I}'_w \subseteq \mathfrak{I}_v$, and consequently $\mathfrak{I}'_w \subseteq (\mathfrak{I}_w \cap \mathfrak{I}_v)$—see Lines 2 and 6 of the Update procedure (Algorithm 4). Hence, the pool-function in the new recursive call Δ-s-BronKerbosch$(P', R' = (C \cup \{v\}, \mathfrak{I}_v), X', B')$ fulfills the claimed condition. □

Next, we show that R contains time-maximal Δ-s-plexes. We further show that the sets P and X contain all time-maximal vertex-interval-set pairs, which can be added to R such that the result still remains a time-maximal Δ-s-plex.

Lemma 7.5. *In each recursive call of* Δ-s-BronKerbosch$(P, R = (C, \mathfrak{I}), X, B)$ *the following holds:*

1. *for all* $I \sqsubseteq \mathfrak{I}$ *it holds that* (C, I) *is a time-maximal* Δ*-*s*-plex,*
2. *for all* $(v, \mathfrak{I}_v) \in P \cup X$ *it holds that for all* $I_v \sqsubseteq \mathfrak{I}_v$, $(C \cup \{v\}, I_v)$ *is a time-maximal* Δ*-*s*-plex, and*
3. *all vertex-interval-set pairs* $(v, \mathfrak{I}_v)$ *satisfying the second property are contained in either* P *or* X*.*

Proof. All properties can be proven by induction on the recursion depth, that is, the number $|C|$ of vertices of the Δ-s-plex in the current recursive call.

Initially, Δ-s-BronKerbosch (Algorithm 2) is called with $R = (C, \mathfrak{I}) = (\emptyset, [\ell - \Delta + 1])$, $P = \{(v, [\ell - \Delta + 1]) \mid v \in V\}$, and $X = \emptyset$. We have that $(\emptyset, [\ell - \Delta + 1])$ is a trivial time-maximal Δ-s-plex. For all $(v, \mathfrak{I}_v) \in P \cup X$ it holds that $\mathfrak{I}_v = [\ell - \Delta + 1]$ and, thus, that $(\emptyset \cup \{v\}, [\ell - \Delta + 1])$ is a time-maximal Δ-s-plex. Obviously, $P \cup X$ contains all vertex-interval-set pairs that form time-maximal Δ-s-plexes with R. Hence, the induction base case holds.

Now let us assume that for a recursive call Δ-s-BronKerbosch$(P, R = (C, \mathfrak{I}), X, B)$ all properties stated in Lemma 7.5 hold. Let $(v, \mathfrak{I}_v) \in P$ be a vertex-interval-set-pair added to R (Lines 8 and 9 of Algorithm 2), that is, $R' = (C', \mathfrak{I}_v)$ with $C' = C \cup \{v\}$. By induction hypothesis, for all $I \sqsubseteq \mathfrak{I}_v$ it holds that (C', I) is a time-maximal Δ-s-plex. Hence, we have that the first property of Lemma 7.5 holds. It remains to show that P and X are suitably adapted for the new recursive call on R'.

We show that P' and X' satisfy the above properties after a call of the Update procedure (Algorithm 4). The Update procedure takes as input the set P (or X) of vertex-interval-set pairs, the vertex set C' of the current Δ-s-plex set R', a set Crit of critical vertex-interval-set pairs, and the newly added vertex-interval-set pair $(v, \mathfrak{I}_v)$. The set Crit contains all vertex-interval-set pairs $(w, \mathfrak{I}_{\text{Crit}})$ such that $\mathfrak{I}_{\text{Crit}} \subseteq \mathfrak{I}_v$ and for all $i \in \mathfrak{I}_{\text{Crit}}$ we have that the vertex w has s Δ-non-neighbors in C' in W_i^Δ, that is, $|\overline{N}^\Delta(w, i) \cap C'| = s$ (Lines 5 to 9).

We now show that Update works as intended for P. The case for X is analogous. The set P' is created as follows: For each $(w, \mathfrak{I}_w) \in P$ with $w \neq v$, the Update procedure "reduces" the interval set $\mathfrak{I}_w$ to $\mathfrak{I}_v$, that is, $\mathfrak{I}'_w = \mathfrak{I}_w \cap \mathfrak{I}_v$. Now, all Δ-windows W_i^Δ with $i \in \mathfrak{I}'_w$, are removed in Lines 5 to 7 of the Update procedure (Algorithm 4) for which

1. vertex w has already s Δ-non-neighbors in C', that is, $|\overline{N}^\Delta(w, i) \cap C'| = s$ (recall that $w \in \overline{N}^\Delta(w, i)$), or
2. vertex w has a non-neighbor in C' that has already s Δ-non-neighbors in C', that is, there exists a vertex $c \in C'$ with $c \in \overline{N}^\Delta(w, i)$ and $|\overline{N}^\Delta(c, i) \cap C'| = s$.

In both cases, w cannot be added to C'. In the end, all starting time steps of the remaining Δ-windows are collected (Line 6). Hence, the second property of Lemma 7.5 holds. By induction hypothesis all intervals $I \sqsubseteq \mathfrak{I}_w$ were time-maximal with respect to $C \cup \{w\}$ and all intervals that form a time-maximal Δ-s-plex with $C \cup \{w\}$ were contained in $\mathfrak{I}_w$. Adding a vertex v to a Δ-s-plex only reduces the size of the set of possible Δ-windows of an additional vertex w. It follows that also the third property of Lemma 7.5 holds and all vertex-interval-set pairs in P' (and X') are time-maximal and complete with respect to C'.

Thus, the recursive call Δ-s-BronKerbosch$(P', R' = (C', \mathfrak{I}_v), X', B')$ fulfills the conditions stated in Lemma 7.5. □

We are now ready to prove the correctness of Δ-s-BronKerbosch.

Proposition 7.6. *For any given temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, Δ-s-BronKerbosch computes all maximal Δ-s-plexes of $\mathcal{G}$.*

Proof. Let $R^\star = (C^\star, I^\star)$ be a maximal Δ-s-plex. We first show that there will be a recursive call adding $R^\star$ to the solution. Since we are building Δ-s-plexes bottom up, there will be a recursive call of Δ-s-BronKerbosch (Algorithm 2) on (P, R, X, B) with $R = (C, \mathfrak{I})$, $C \subseteq C^\star$, $I^\star \subseteq \mathfrak{I}$ and $|C| = |C^\star| - c$ for all $c = 0, 1, \ldots, |C^\star|$. Additionally, all vertices $v \in C^\star \setminus C$ with $I^\star \subseteq \mathfrak{I}_v$, called *candidates*, will be contained in P. We show this by induction on $|C|$.

Clearly, in the initial call, $C = \emptyset \subseteq C^\star$ and $I^\star \subseteq [\ell - \Delta + 1]$. Since $P = \{(v, [\ell - \Delta + 1]) \mid v \in V\}$, every vertex $v \in C^\star$ is contained in P. Hence, the induction base case holds.

Now assume that there is a recursive call with (P, R, X, B), where $R = (C, \mathfrak{I})$, $C \subseteq C^\star$, $I^\star \subseteq \mathfrak{I}$, and all candidates are contained in P. Consider the first candidate $(v, \mathfrak{I}_v)$ with $v \in C^\star$ in the for-loop (Line 7 of Algorithm 2) of that recursive call. After adding v to C, since $R^\star$ is a Δ-s-plex, according to Lemma 7.5 all other candidates are still contained in P' after a call of Update (Algorithm 4). Since $(v, \mathfrak{I}_v)$ was a candidate, it holds for the new Δ-s-plex set $R' = (C' = C \cup \{v\}, \mathfrak{I}_v)$ that $C' \subseteq C^\star$ and $I^\star \subseteq \mathfrak{I}_v$. Hence, by induction, there is a recursive call with $R = (C^\star, \mathfrak{I}^\star)$ with $I^\star \sqsubseteq \mathfrak{I}^\star$ and, since $R^\star$ is maximal, there is no vertex-interval-set pair $(v, \mathfrak{I}) \in P \cup X$ with $I^\star \subseteq \mathfrak{I}$. Thus, $(C^\star, I^\star)$ is enumerated in Line 4 of Algorithm 2.

Now assume that some pair (C, I) is added to the solution in Line 4 of Algorithm 2. We show that (C, I) is a maximal Δ-s-plex. By Lemma 7.5 we know that (C, I) is a time-maximal Δ-s-plex and we know that all vertex-interval-set pairs $(v, \mathfrak{I}_v)$, where $R' = (C \cup \{v\}, \mathfrak{I}_v)$ is a set of Δ-s-plexes, are contained in $P \cup X$. Since we check whether $\forall (w, \mathfrak{I}_w) \in P \cup X$ and $\forall I_w \sqsubseteq \mathfrak{I}_w \colon I_w \neq I$ in Line 3 of Algorithm 2, it follows that there

is no vertex in P or X which can be added without decreasing the interval I, hence, (C, I) is also vertex-maximal. Thus, (C, I) is a maximal Δ-s-plex. □

7.3.2 Running Time of Δ-s-BronKerbosch

We have shown that Δ-s-BronKerbosch (Algorithm 2) enumerates all maximal Δ-s-plexes in a temporal graph. In this section, we analyze its running time in four steps. First, we determine the running time of precomputing the Δ-non-neighborhoods. Then, we analyze the running time of UpdatePool (Algorithm 3) and Update (Algorithm 4). In a third step, we prove an upper bound on the number of time-maximal Δ-s-plexes that depends on the Δ-slice degeneracy of the temporal graph and show that there is at most one recursive call for each of them. Finally, we combine our findings to obtain an upper bound on the running time of Δ-s-BronKerbosch.

The running time of computing the Δ-non-neighborhood has a big influence on the overall running time since it is accessed multiple times in each recursive call. However, it can be precomputed once before the initial call of Δ-s-BronKerbosch. In the following lemma, we show an upper bound on the running time of this computation assuming that the edges are sorted by their time stamps.

Lemma 7.7. *If the edges are sorted by their time stamps, then the Δ-non-neighborhood for all vertices over the whole lifetime, that is, $\overline{N}^{\Delta}(v, [\ell - \Delta + 1])$ for all $v \in V$, can be computed in $O(|V|^2 + \sum_{i=1}^{\ell} |E_i|)$ time.*

Proof. First, for each pair of vertices v, w we initially set their Δ-non-neighborhood to the whole lifetime of the temporal graph. The initialization can be done in $O(|V|^2)$ time. Then, for each time step $t \in [\ell]$ and each edge $\{v, w\} \in E_t$ the Δ-neighborhood interval $[t - \Delta + 1, t]$ is cut out of the Δ-non-neighborhood of v and of w. Due to the sorting of the edges by time stamps, this can be done in $O(\sum_{i=1}^{\ell} |E_i|)$ time. We end up with a sorted list of Δ-non-neighborhood intervals for each vertex pair with at most $O(\min\{\sum_{i=1}^{\ell} |E_i|, \ell\})$ many non-overlapping time intervals. In the last step, the Δ-non-neighborhood $\overline{N}^{\Delta}(v, [\ell - \Delta + 1])$ for each vertex v is computed in $O(|V|^2 + \sum_{i=1}^{\ell} |E_i|)$ time using the sorted lists of Δ-non-neighborhood intervals of all vertex pairs containing v. □

Next, we determine the running time of UpdatePool and Update. To do that, we first show how we compute unions, intersections, and differences of sets of intervals.

Lemma 7.8. *Let $\mathfrak{I}$ and $\mathfrak{J}$ be two sets of intervals appearing at some computation step of Δ-s-BronKerbosch that are both sorted by the starting points of the intervals. Then $\mathfrak{I} \cup \mathfrak{J}$, $\mathfrak{I} \cap \mathfrak{J}$, and $\mathfrak{I} \setminus \mathfrak{J}$ can all be computed in $O(\min\{\sum_{i=1}^{\ell} |E_i|, \ell\})$ time. such that the outcoming interval sets are sorted by the starting points of the intervals.*

Proof. We first discuss how we store sets of intervals: Given an interval set $\mathfrak{I}$ we store the list of its "maximal" intervals I, that is, all $I \sqsubseteq \mathfrak{I}$. These intervals can be represented by their starting points and end points. The list of intervals is stored ordered by the starting points of the intervals. Furthermore, each interval in an interval set is induced by a different time-stamped edge. Hence, the size of each set of intervals can be bounded by $O(\min\{\sum_{i=1}^{\ell} |E_i|, \ell\})$. We briefly discuss how we compute $\mathfrak{I} \cup \mathfrak{J}$, $\mathfrak{I} \cap \mathfrak{J}$, and $\mathfrak{I} \setminus \mathfrak{J}$ efficiently given two interval sets $\mathfrak{I}$ and $\mathfrak{J}$ in the above described representation. We claim that each set operation can be performed $O(\min\{\sum_{i=1}^{\ell} |E_i|, \ell\})$ time in the following way such that the output interval set is also in the described representation.

1. Select the first interval I and J from $\mathfrak{I}$ and $\mathfrak{J}$, respectively. Set $I^\star = \emptyset$ (only needed for the computation of $\mathfrak{I} \cup \mathfrak{J}$).

2. Depending on which operation should be computed, do one of the following:

   $\mathfrak{I} \cup \mathfrak{J}$: If $I^\star = \emptyset$ then set $I^\star = I$, if the startpoint of I is smaller than the startpoint of J, and set $I^\star = J$ otherwise. If $I \cup J = [a, b]$ for some $a, b \in \mathbb{N}$, that is, $I \cup J$ is also an interval, then set $I^\star = I^\star \cup I \cup J$, otherwise, if $I^\star \neq \emptyset$, then add $I^\star$ to the output interval set and set $I^\star = \emptyset$. If the endpoint of I is smaller than the endpoint of J, then replace I with the next interval in $\mathfrak{I}$, otherwise replace J with the next interval in $\mathfrak{J}$.

   $\mathfrak{I} \cap \mathfrak{J}$: If $I \cap J \neq \emptyset$, then add $I \cap J$ to the output interval set. If the endpoint of I is smaller than the endpoint of J, then replace I with the next interval in $\mathfrak{I}$, otherwise replace J with the next interval in $\mathfrak{J}$.

   $\mathfrak{I} \setminus \mathfrak{J}$: If $I \setminus J = I$, then add I to the output interval set. Otherwise, if $I \setminus J = [a, b]$ for some $a, b \in \mathbb{N}$, that is, $I \setminus J$ is also an interval, then set $I = I \setminus J$. If $I \setminus J = [a, b] \cup [a', b']$ with $b < a'$ for some $a, b, a', b' \in \mathbb{N}$, that is, $I \setminus J$ is composed of two intervals, then add $[a, b]$ to the output interval set and set $I = [a', b']$. If the endpoint of I is smaller than the endpoint of J, then replace I with the next interval in $\mathfrak{I}$, otherwise replace J with the next interval in $\mathfrak{J}$.

3. Repeat Step 2 until all intervals in $\mathfrak{I}$ and $\mathfrak{J}$ are processed.

It is easy to verify that the above procedure runs in $O(\min\{\sum_{i=1}^{\ell} |E_i|, \ell\})$ time for each set operation. □

Now we are ready to analyze the running time of UPDATEPOOL and UPDATE.

Lemma 7.9. *The procedures* UPDATEPOOL *and* UPDATE *run in* $O(\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}\cdot|V|^2)$ *time.*

Proof. First, let us briefly discuss the structure of the pool function. In the pool function B, we store for each vertex v and each Δ-window W_t^Δ the number of Δ-non-neighbors in the current Δ-s-plex. This information can be stored for each vertex in the following way. For each vertex v, we can store a set of integer-interval pairs (i_v, I_v) such that for all $t \in I_v$ we have that $B(v,t) = i_v$. Note that for a vertex v, each change in the number of Δ-non-neighbors in the current Δ-s-plex is induced by a time-stamped edge between v and a vertex in the current Δ-s-plex. Hence, the number of different integer-interval pairs for each vertex v in B is bounded by $O(\min\{\sum_{t\in[\ell]}|\{\{v,w\}\mid\{v,w\}\in E_t\}|,\ell\})$. In the following argument, we make use of Lemma 7.8 whenever unions, intersections or differences of sets of intervals are computed.

We first analyze the running time of the UPDATEPOOL (Algorithm 3) procedure. Initializing Crit and B' in Lines 2 and 3 takes $O(|V|)$ and $O(\min\{\sum_{i=1}^{\ell}|E_i|,|V|\ell\})$ time, respectively. Next, for each $(w,\mathfrak{I}_w)\in P\cup X\cup\{(c,\mathfrak{I}_v)\mid c\in C\}$ (Line 4) we compute the cut with the Δ-non-neighborhood $\overline{N}^\Delta(v,\mathfrak{I}_v)$ (Line 5) in $O(|V|\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\})$ time. Updating the pool function (Lines 6 and 7) also takes $O(\min\{\sum_{i=1}^{\ell}|E_i|,\ell\})$ time. Filtering the critical time intervals (Lines 8 and 9) can be done during the update with no extra time consumption. There are $|V|$ vertex-interval-set pairs in $P\cup X\cup\{(c,\mathfrak{I}_v)\mid c\in C\}$. Hence, the overall time is in $O(|V|\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}))$.

Next, consider the UPDATE (Algorithm 4) procedure. Reducing P to the interval set $\mathfrak{I}_V$ (Line 2) can be done in $O(|V|\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\})$ time. There are at most $|V|$ elements in the set P_{reduced}. For each $(w,\mathfrak{I}_w)\in P_{\text{reduced}}[V(P)\setminus\{v\}]$ (Line 4), we compute the cut $\text{Crit}[C\cup\{w\}]\sqcap\overline{N}^\Delta(w,\mathfrak{I}_w)$ (Line 5) in $O(|V|\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\})$ time. In this cut, there are at most $O(|V|\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}$ vertex-interval-set pairs. For each of these elements $(u,\mathfrak{I}_u)$ we can compute $\mathfrak{I}_w\setminus\mathfrak{I}_u$ (Line 6) in $O(\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}$ time. Altogether, the running time of this whole procedure is in $O(|V|^2\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\})$. □

Moving forward, we now upper-bound the number of recursive calls of Δ-s-BRON-KERBOSCH.

Lemma 7.10. *For each time-maximal* Δ-s-*plex* (C,I) *of a temporal graph* $\mathcal{G}$*, there is at most one recursive call of* Δ-s-BRONKERBOSCH *with* $R=(C,\mathfrak{I})$ *with* $I\sqsubseteq\mathfrak{I}$ *as input.*

Proof. Assume that there are two recursive calls $\mathscr{A}$ and $\mathscr{B}$ of Δ-s-BRONKERBOSCH with $R_\mathscr{A}=(C,\mathfrak{I}_\mathscr{A})$ with $I\sqsubseteq\mathfrak{I}_\mathscr{A}$, and $R_\mathscr{B}=(C,\mathfrak{I}_\mathscr{B})$ with $I\sqsubseteq\mathfrak{I}_\mathscr{B}$, respectively, as part of the input. Let $R^\star=(C^\star,\mathfrak{I}^\star)$ with $C^\star\subset C$ and $\mathfrak{I}_\mathscr{A}\subseteq\mathfrak{I}^\star$, $\mathfrak{I}_\mathscr{B}\subseteq\mathfrak{I}^\star$, be in the input

of the least common ancestor of $\mathscr{A}$ and $\mathscr{B}$ in the tree of recursive calls. Let $P^\star$ be the candidate set of that recursion call. There must be two vertex-interval-set pairs $(v,\mathfrak{I}_v),(w,\mathfrak{I}_w) \in P^\star$ that lead to the recursive calls $\mathscr{A}$ and $\mathscr{B}$, respectively.

Clearly, for all $I \sqsubseteq \mathfrak{I}$ it holds that $I \subseteq \mathfrak{I}_v$ and $I \subseteq \mathfrak{I}_w$, and we have that $\{v,w\} \subseteq C$. Since for a fixed vertex all intervals in a candidate set are distinct, as shown in Lemma 7.5, it follows that $v \neq w$. Without loss of generality, assume that $(v,\mathfrak{I}_v)$ is considered first in the for-loop over all candidates in $P^\star$ (Line 8 of Algorithm 2) and its selection leads to recursive call $\mathscr{A}$. After $(w,\mathfrak{I}_w)$ is considered, the vertex-interval-set pair $(v,\mathfrak{I}_v)$ is moved to $X^\star$ and removed from $P^\star$. Hence, all following recursive calls do not consider $(v,\mathfrak{I}'_v)$ for any $\mathfrak{I}'_v$. Recall that we assumed that the recursive call $\mathscr{B}$ outputs $R = (C,\mathfrak{I}_{\mathscr{B}})$. Since for all $I \sqsubseteq \mathfrak{I}$ it holds that $I \subseteq \mathfrak{I}_v$ and $v \in C$, it follows that a vertex-interval-set pair $(v,\mathfrak{I}'_v)$ with the property that

a) for all $I \sqsubseteq \mathfrak{I}$, it holds that $I \subseteq \mathfrak{I}'_v$ and

b) for all $I' \sqsubseteq \mathfrak{I}'_v$, it holds that $I' \subseteq \mathfrak{I}_v$

needs to be considered in a future call. This contradicts the fact that we do not consider vertex-interval-set pairs that are contained in X. Thus, there cannot be two recursive calls of Δ-s-BronKerbosch with the same C and it follows that for each time-maximal Δ-s-plex (C,I) there is at most one recursive call of Δ-s-BronKerbosch with $R = (C,\mathfrak{I})$ with $I \sqsubseteq \mathfrak{I}$ as input. □

Finally, we upper-bound the number of time-maximal Δ-s-plexes in a temporal graph $\mathscr{G} = (V,(E_i)_{i\in[\ell]})$ using the Δ-slice degeneracy value d (Definition 7.7) of $\mathscr{G}$.

Proposition 7.11. *Let $\mathscr{G} = (V,(E_i)_{i\in[\ell]})$ be a temporal graph with Δ-slice degeneracy d. The number of time-maximal Δ-s-plexes in $\mathscr{G}$ is at most $|V| \cdot \binom{|V|}{s-1} \cdot 2^{d+s} \cdot \min\{\sum_{i=1}^{\ell} |E_i|, \ell\}$.*

Proof. The statement can be shown by a simple counting argument. For each Δ-window, we count how many time-maximal Δ-s-plexes have a lifetime that contains this Δ-window. For a given Δ-window W_i^Δ, there exists a *degeneracy ordering* of $G_{W_i^\Delta} = (V,E_{W_i^\Delta})$, where $E_{W_i^\Delta} = \{e \mid e \in E_t \wedge t \in W_i^\Delta\}$. The degeneracy ordering of a graph is a linear ordering of its vertices with the property that for each vertex v at most d of its neighbors occur at a later position.

Now for each Δ-window W_i^Δ with the property that E_{Δ_i} contains a unique set of time edges and for each vertex v (that is, $\min\{\sum_{i=1}^{\ell} |E_i|, \ell\} \cdot |V|$ possibilities) we consider the degeneracy ordering of graph $G_{\Delta_i} = (V,E_{\Delta_i})$. We count the number of s-plexes of G_{Δ_i} which only contain v and vertices that appear at a later position in the ordering. By definition, v has at most d neighbors that appear later in the ordering

and v can be connected to $s-1$ other vertices. For the latter, we consider all vertices, yielding the factor $\binom{|V|}{s-1}$ in the upper bound. Each subset of these $d+s$ vertices in G_{Δ_i} can, together with v, potentially be the vertex set of several Δ-s-plexes. The number of such subsets is at most 2^{d+s}. This yields our upper bound since each of these vertex sets can potentially form at most one *time-maximal* Δ-s-plex with a lifetime that contains Δ-window W_i^Δ. Putting all pieces together, we arrive at the claimed upper bound on the number of time-maximal Δ-s-plexes in a temporal graph. □

We now combine the previous results to upper-bound the running time of Δ-s-BronKerbosch (Algorithm 2).

Proposition 7.12. *Δ-s-BronKerbosch runs in $O(\binom{|V|}{s-1}\cdot 2^{d+s}\cdot\min\{(\sum_{i=1}^{\ell}|E_i|)^2,\ell^2\}\cdot|V|^3)$ time, where d is the Δ-slice degeneracy of the input graph.*

Proof. First, recall that by Lemma 7.7 the Δ-non-neighborhood can be precomputed once at the start of the algorithm in $O(|V|^2+\sum_{i=1}^{\ell}|E_i|)$ time assuming that the edges are sorted. If they are unsorted, then we can sort them in $O(\sum_{i=1}^{\ell}|E_i|\cdot\log\sum_{i=1}^{\ell}|E_i|)$ time.

We first give an upper bound on the number of recursive calls in an execution of Δ-s-BronKerbosch. By Lemma 7.5(1) we know that in each recursive call of Δ-s-BronKerbosch, we have that (C,I) is a time-maximal Δ-s-plex for all $I\sqsubseteq\mathfrak{I}$. Lemma 7.10 tells us that the time-maximal Δ-s-plexes in all recursive calls are distinct. Finally, Proposition 7.11 gives us an upper bound on the number of distinct time-maximal Δ-s-plexes. We can conclude that for an execution of Δ-s-BronKerbosch the number of recursive calls is bounded by $O(\binom{|V|}{s-1}\cdot 2^{d+s}\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}\cdot|V|)$.

Now, we analyze the running time of each recursive call. For this, notice that there is exactly one call to UpdatePool and two calls to Update in each recursive call of Δ-s-BronKerbosch. Hence, the running time of the for-loop is dominated by the complexity of UpdatePool and Update which run in $O(\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}\cdot|V|^2)$ time by Lemma 7.9. Concluding the proof, there are $O(\binom{|V|}{s-1}\cdot 2^{d+s}\cdot\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}\cdot|V|)$ recursive calls and each of these recursive calls runs in $O(\min\{\sum_{i=1}^{\ell}|E_i|,\ell\}\cdot|V|^2)$ time. This yields a total running time of $O(\binom{|V|}{s-1}\cdot 2^{d+s}\cdot\min\{(\sum_{i=1}^{\ell}|E_i|)^2,\ell^2\}\cdot|V|^3)$ for Δ-s-BronKerbosch. □

Theorem 7.3 now follows immediately from Proposition 7.6 and Proposition 7.12.

7.4 Conclusion

In this chapter, we introduced a temporal adaptation of s-plexes and adapted the classic Bron-Kerbosch algorithm for enumerating all maximal Δ-s-plexes in a

temporal graph. We studied its running time, showing that TEMPORAL CLIQUE and TEMPORAL s-PLEX for all $s \geq 1$ are fixed-parameter tractable when parameterized by the Δ-slice degeneracy of the input temporal graph.

We remark that the papers this chapter is based on [Ben+19, Him+17] also contain an implementation and an experimental evaluation of the proposed algorithm. In experiments on real-world networks, it was shown that our algorithm performs better than the state-of-the-art algorithm by Viard, Magnien, and Latapy [VML18] on most instances but is also heavily out-performed on one of our instances. Moreover, Bentert et al. [Ben+19] also measured the Δ-slice degeneracy of many real-world instances showing that it is indeed small. Hence, our parameterized complexity analysis of this problem gives a sound theoretical explanation for the good practical performance of our algorithm.

The experiments of Bentert et al. [Ben+19] also suggest that the number of trivial solutions[16] for increasing s greatly limits the scalability of any algorithm enumerating all maximal Δ-s-plexes. Thus, they proposed to instead enumerate all maximal *connected* s-plexes of minimum order $2s+1$. This allowed them to design heuristics to speed up their algorithm in practice.

We believe that in the context of TEMPORAL CLIQUE and TEMPORAL s-PLEX, restricting the input to temporal unit interval graphs (as discussed in Chapter 4 in the context of TEMPORAL (s,z)-SEPARATION) would be an interesting starting point to develop tailored algorithms for clique enumeration in physical proximity networks. The goal would be to exploit additional structure that the network model provides to obtain faster algorithms. Notably, Δ-cliques were originally introduced in the context of analyzing physical proximity networks [VLM16].

[16] Any vertex set of size s is a trivial s-plex.

CHAPTER 8

Temporal Cluster Editing

Motivated by the recent rapid growth of research for algorithms to cluster temporal graphs, we study extensions of the classic CLUSTER EDITING problem. In TEMPORAL CLUSTER EDITING we aim to transform all layers of a temporal graph into cluster graphs (disjoint unions of cliques) such that the resulting clusterings "look similar". we want to mark at most d vertices and to transform each layer into a cluster graph using at most k edge additions or deletions per layer so that, if we remove the marked vertices, then we obtain the same cluster graph in all layers. We study the combinatorial structure of this problem and fully classify its parameterized complexity with respect to the parameters d, k, the number of vertices, and the lifetime ℓ of the input temporal graph. Among other things, we show that TEMPORAL CLUSTER EDITING is fixed-parameter tractable when parameterized by $k+d$ and admits a polynomial kernel when parameterized by $k+d+\ell$.

This chapter is based on the paper "Cluster Editing in Multi-Layer and Temporal Graphs" by Chen et al. [Che+18].

8.1 Introduction

CLUSTER EDITING and its weighted form CORRELATION CLUSTERING are two important and well-studied models of graph clustering [BB13, BBC04, SST04]. In the former, we are given a graph and we aim to edit (that is, add or delete) the fewest number of edges in order to obtain a *cluster graph*, a graph in which each connected component is a clique. CLUSTER EDITING has attracted a lot of attention from a parameterized-algorithms point of view [BB13, BFK18, CC12, Fom+14, Gra+05, KU12, Luo+18] and many of the resulting contributions have found their way back into practice [BB13, Section 6].

However, in many application areas additional information is available and used in clustering methods. In particular, research on clustering multi-layer and temporal graphs grows rapidly [KL15, TAG17, TB11, TBK07, TLD09]. In the context of multi-layer networks, a layer can represent social interactions, geographic closeness, common interests or activities [KL15].

The goals in clustering multi-layer and temporal graphs are, respectively, to find a clustering that is consistent with all layers [Kiv+14, KL15, TAG17, TLD09] or a

clustering that slowly evolves over time consistently with the graph [TB11, TBK07]. The methods used herein are often heuristic and, beyond observing NP-hardness, to the best of our knowledge, there is no deeper analysis of the complexity of the general underlying computational problems that are attacked in this way. Hence, there is also a lack of knowledge about the possible avenues for algorithmic tractability. We initiate this research here.

We analyze the combinatorial structure behind cluster editing for temporal graphs via studying the parameterized complexity with respect to the most basic parameters, such as the number of edits.

Informally, we model Temporal Cluster Editing as follows. We receive an input temporal graph and aim to transform all layers into cluster graphs that differ only slightly. We use a very basic approach to model this: We want to mark at most d vertices and to transform each layer into a cluster graph using at most k edge additions or deletions per layer so that, if we remove the marked vertices, we obtain the same cluster graph in all layers. We believe that this is a canonical way to allow "small changes over time" in the clustering. An alternative would be to only require the clusterings of *adjacent* layers to be similar. This model has been studied by Chen et al. [Che+18] and even though it might be better suited than our model for certain applications, this alternative model seems to be computationally much harder [Che+18].

As we will see, our problem offers rich interactions between the layers on top of the structure inherited from Cluster Editing. Our main contributions are an intricate fixed-parameter algorithm for Temporal Cluster Editing, whose underlying techniques should be applicable to a broader range of temporal and multi-layer graph problems, and a polynomial kernel for Temporal Cluster Editing for a larger parameter combination.

8.1.1 Related Work

In the static setting Cluster Editing has been thoroughly investigated [BB13, BBC04, BFK18, CC12, Fom+14, Gra+05, KU12, SST04].

We remark that Chen et al. [Che+18] also study another model for temporal cluster editing.[17] In that model, also each layer is transformed into a cluster graph but rather than marking vertices that may change clusters, the cluster graphs may change over time. In particular, this means that the cluster graph of the first and the last layer may look very different if the temporal graph has a sufficiently long lifetime.

In terms of parameterized algorithms, only the indirect approach of aggregating

[17] The model we study in this chapter is called Multi-Layer Cluster Editing by Chen et al. [Che+18].

clusterings into one has been studied for multi-layer [Bet+11, Dör+14] and temporal graphs [TB11, TBK07]. The approximability of temporal versions of k-means clustering and its variants was studied by Dey, Rossi, and Sidiropoulos [DRS17]. A dymanic version of cluster editing has been studied by Luo et al. [Luo+18]. In the dynamic setting a given cluster graph for a first input graph should be transformed into a "similar" cluster graph for a second input graph.

8.1.2 Our Contributions and Organization of the Chapter

We completely classify Temporal Cluster Editing in terms of fixed-parameter tractability and existence of polynomial-size problem kernels with respect to the parameters "number d of marked vertices", "number k of edge modifications per layer", "number $|V|$ of vertices", and "number ℓ of layers", and all of their combinations, see Figure 8.1 for an overview. Temporal Cluster Editing is para-NP-hard (NP-hard for constant parameter values) for all parameter combinations which are smaller or incomparable to $k+d$. While it is known that the problem is NP-complete even if both $d=0$ and $\ell=1$ or both $k=0$ and $\ell=3$ [Che+18], we show that Temporal Cluster Editing is polynomial-time solvable if $k=0$ and $\ell\leq 2$. Finally, we show that Temporal Cluster Editing admits a polynomial kernel with respect to the parameter combination $d+k+\ell$ and does not admit a polynomial kernel for the parameter "number n of vertices" unless NP $\subseteq$ coNP/poly.

The chapter is organized as follows. In Section 8.2 we formally introduce all necessary concepts related to (temporal) cluster editing that we need in this chapter, we formally define our problem setting, and we report some basic observations about the problem. In Section 8.3 we present our main FPT-algorithm for Temporal Cluster Editing. In Section 8.4 we present a polynomial kernel for Temporal Cluster Editing. We conclude in Section 8.5.

8.1.3 Further Contributions of the Paper this Chapter is Based on

Additionally to the contributions we present in this chapter, Casteigts et al. [Cas+20] show that Temporal Cluster Editing is NP-complete even if $k=0$ and $\ell\geq 3$.

Chen et al. [Che+18] further studied an alternative model for temporal cluster editing where between adjacent time steps only few vertices may change their clusters. They show that this problem is W[1]-hard when parameterized by the number k of modifications per layer even if only three vertices may change cluster when going from one time step to the next. On the positive side they showed that this problem can be solved in polynomial time if k is constant and that the polynomial kernel for Temporal Cluster Editing can be adapted to this alternative model.

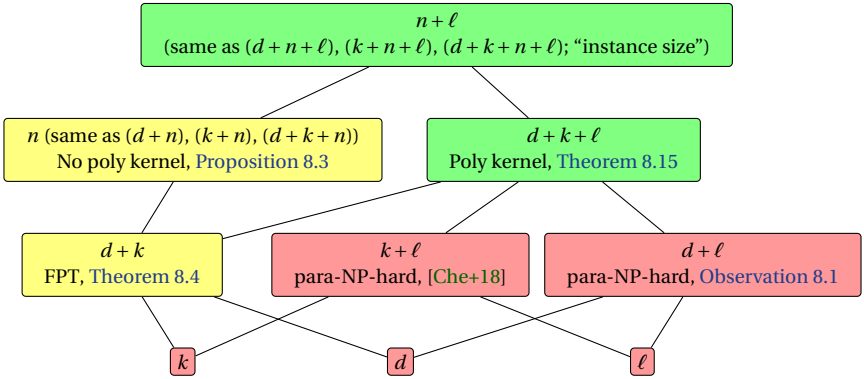


Figure 8.1: Our results (and to give a complete picture, one result from Chen et al. [Che+18]) for TEMPORAL CLUSTER EDITING in a Hasse diagram of the upper-boundedness relation between the parameters "number k of edge modifications per layer", "number d of marked vertices", "number ℓ of layers", and "number $n = |V|$ of vertices" and all of their combinations. Red entries mean that TEMPORAL CLUSTER EDITING parameterized by the parameter in the entry is para-NP-hard (NP-hard for constant parameter values). It is in FPT for all parameter combinations colored yellow or green and admits a polynomial kernel for all parameter combinations colored green. For all parameter combinations that are colored yellow, it does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.

8.2 Preliminaries

In this section, we formally introduce the most important concepts related to static and temporal cluster editing and give the formal problem definition of TEMPORAL CLUSTER EDITING. We further discuss some basic observations for TEMPORAL CLUSTER EDITING.

8.2.1 Static Cluster Editing

We call a static graph $G = (V, E)$ a *cluster graph* if every connected component of G is a clique. Alternatively, we can define a cluster graph as a graph that does not contain an induced P_3, where a P_3 is a path on three vertices, that is, it is isomorphic to a graph with three vertices and two edges.

The NP-complete CLUSTER EDITING problem asks whether it is possible to transform a given graph into a cluster graph by performing at most k edge modifications, that is, adding or removing edges [BBC04, SST04]. We formally define this as follows.

An *edge modification* or *edit* for a graph $G = (V, E)$ is an unordered pair of vertices

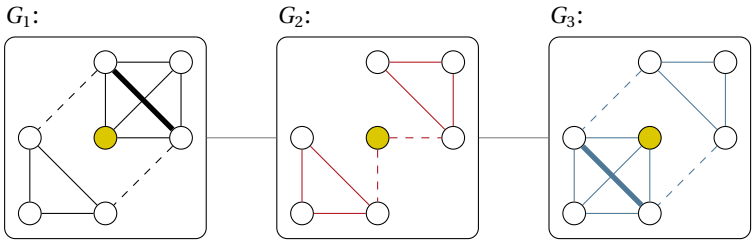


Figure 8.2: Example temporal graph with lifetime three and with a 1-consistent 3-bounded clustering. The marked vertex is colored in yellow. The modification sets consist of thick and dashed edges, where thick edges are added and dashed edges are removed.

from V. Let M be a set of edits for G. If the graph $G' = (V, E \oplus M)$ is a cluster graph, then we say that M is a *cluster editing set* for G. Herein, $\oplus$ denotes the symmetric difference: $A \oplus B = (A \setminus B) \cup (B \setminus A)$.

8.2.2 Temporal Cluster Editing

To transfer Cluster Editing to the temporal setting, we propose the following. We want to transform every layer of the input temporal graph by performing at most k edge modifications per layer. We call a clustering produced in this way *k-bounded*. However, without any further restrictions the resulting cluster graphs can look vastly different and we are essentially solving ℓ independent Cluster Editing instances. Since we want that there are no big changes in the clustering over time, we introduce a set of "marked vertices" into our model. These vertices may change their clusters over time, but the rest of the cluster graphs must stay the same. If we have to mark at most d vertices in a clustering to achieve this, then we call the clustering *d-consistent*. We give an example in Figure 8.2. Formally, we define these concepts as follows.

Definition 8.1 (d-Consistent k-Bounded Clustering)**.** Let $\mathcal{G} = (V, (E_i)_{i \in [\ell]})$ be a temporal graph. A *clustering* for $\mathcal{G}$ is a sequence $\mathcal{M} = (M_i)_{i \in [\ell]}$ of edge modification sets such that M_i is a cluster editing set for layer G_i. For that, we say that $\mathcal{M}$ is *k-bounded* for some integer $k \in \mathbb{N}$ if $|M_i| \leq k$ for each $i \in [\ell]$. Let $G'_i = (V, E_i \oplus M_i)$ for all $i \in [\ell]$. Clustering $\mathcal{M}$ is *d-consistent* for some integer $d \in \mathbb{N}$ if there is a single subset $D \subseteq V$ of vertices with $|D| \leq d$ such that $G'_i[V \setminus D] = G'_j[V \setminus D]$ for all $i, j \in [\ell]$. We say that D *witnesses* that $\mathcal{M}$ is d-consistent.

A tuple $(M_1, \ldots, M_\ell, D)$ of edge modification sets and a set of "marked" vertices D is

a *solution* if D witnesses that $\mathcal{M} = (M_i)_{i\in[\ell]}$ is a d-consistent k-bounded clustering.

Intuitively, the sets M_i contain the data that we need to disregard in order to cluster our input and hence we want to minimize their sizes [TB11, TBK07]. The set D contains the vertices that may move around between clusters over time and hence we also want to keep its size small.

Now we are ready to state the main decision problem of this chapter.

Temporal Cluster Editing	
Input:	A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers $k, d \in \mathbb{N}$.
Question:	Is there a d-consistent k-bounded clustering for $\mathcal{G}$?

Temporal Cluster Editing clearly generalizes the NP-complete Cluster Editing problem [BBC04, KM86, SST04] and it is further easy to check that, given a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, it can be verified in polynomial time whether a tuple $(M_1, \ldots, M_\ell, D)$ of edge modification sets and a set of marked vertices D is a solution for $\mathcal{G}$. Hence, we have that Temporal Cluster Editing in contained in NP and thus is NP-complete.

We remark that from a mathematical point of view, Temporal Cluster Editing can be treated as a *multi-layer graph* problem: It is easy to check that, given a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers $k, d \in \mathbb{N}$, it holds that for every permutation $\pi : [\ell] \to [\ell]$ we have that $(\mathcal{G}, k, d)$ is a YES-instance if and only if $(\mathcal{G}' = (V, (E_{\pi(i)})_{i\in[\ell]}), k, d)$ is a YES-instance.

8.2.3 Basic Observations

We now present some observations on the complexity of Temporal Cluster Editing on few layers. In particular, we obtain a complexity dichotomy for Temporal Cluster Editing with $k = 0$ showing that for $\ell \leq 2$ the problem is polynomial-time solvable. Chen et al. [Che+18] showed that Temporal Cluster Editing is NP-complete if $k = 0$ for all $\ell \geq 3$.

Since Cluster Editing is NP-complete [BBC04, KM86, SST04], we immediately get NP-hardness for Temporal Cluster Editing even if $d = 0$.

Observation 8.1. Temporal Cluster Editing *is NP-complete for all* $d \geq 0$ *and* $\ell \geq 1$.

Now, we turn to the scenario where we are not allowed to edit any edges (that is, $k = 0$). We find that for two layers our problem is related to computing a maximum-weight matching in a bipartite graph, which is polynomial-time solvable.

Proposition 8.2. *If* $k = 0$ *and* $\ell = 2$*, then* Temporal Cluster Editing *can be solved in* $O(|V|^2 \log |V|)$ *time.*

Proof. Let $(\mathcal{G} = (V, E_1, E_2), k = 0, d)$ be an input instance of Temporal Cluster Editing. We claim that the following procedure decides in $O(|V|^2 \log |V|)$ time whether there is a subset $D \subseteq V$ of at most d vertices such that $G_1[V \setminus D] = G_2[V \setminus D]$.

1. Check whether G_1 and G_2 are both cluster graphs, if at least one is not, answer NO.
2. Create a complete (edge-weighted) bipartite graph $H = (A \uplus B, E, w : E \to [|V|])$ in the following way:
   a) For each maximal clique X in G_1 add a vertex v_X to A.
   b) For each maximal clique X in G_2 add a vertex v_X to B.
   c) Add an edge between each two vertices $v_X \in A$ and $v_Y \in B$ with edge weight $w(\{v_X, v_Y\}) = |X \cap Y|$.
3. Compute a maximum-weight matching for H. If the weight of the matching is at least $|V| - d$, answer YES, otherwise answer NO.

Running Time. It is well-known that the first step reduces to checking whether there is an induced P_3 in one of the graphs, which can be done in $O(|V| + |E_1| + |E_2|)$ time.[18] The second step can be performed in $O(|V| + |E_1| + |E_2|)$ time as follows. Find all connected components in G_1 and label the vertices in G_1 according to the components that contain them. Add a vertex v_X to A for each label X. The vertices in B are constructed analogously. Now, to compute the edge weights in H, iterate over all vertices in V and add an edge of weight one to H that is incident to the two corresponding vertices or increase the edge weight if the edge was added in a previous iteration. Note that H contains at most n edges. Finally, the third step can be carried out in $O(|V|^2 \log |V|)$ time using the Hungarian algorithm [Kuh55], which also dominates the running time.

Correctness. First, note that if one of G_1 and G_2 is not a cluster graph, then we clearly face a NO-instance, which is correctly identified by the algorithm in the first step. So from now on, let us assume that both G_1 and G_2 are cluster graphs. To show the correctness of the last step, assume that there is a vertex subset $D \subseteq V$ of size at most d such that $G_1[V \setminus D] = G_2[V \setminus D]$. Let $q_1, q_2, \ldots, q_x$ be the maximal cliques remaining in $G_1[V \setminus D]$. One can verify that the following matching M has weight

[18] This can be done using breadth-first search (BFS) roughly in the following way: Start BFS at a vertex v that is not connected to all other vertices in its component (if such vertex does not exist, then the component is already a clique). As soon as BFS reaches a vertex of distance two to v, one has found an induced P_3.

$|V| - |D|$: For each clique q_i, add to M the edge $\{v_X, v_Y\}$ where X and Y are the two cliques that contain q_i in G_1 and G_2, respectively. Note that since G_1 and G_2 are two cluster graphs on the same vertex set, no maximal clique remaining in $G_1[V \setminus D]$ belongs to two different maximal cliques in G_1 or G_2. Thus, M is indeed a matching. It is straightforward to see that it has weight $|V| - |D|$.

In the opposite direction, assume that H admits a matching M with weight at least $|V| - d$. We consider the following subset V' of vertices: For each edge $\{v_X, v_Y\}$ in M, add to V' all vertices in $X \cap Y$, which is the weight of $\{v_X, v_Y\}$ in H. By the definition of the edge weights in H, it follows that $G_1[V'] = G_2[V']$. Thus, if we remove by marking all vertices in $V \setminus V'$, then both cluster graphs become the same. Since M is a matching, it follows $|V'| = w(M) \geq |V| - d$. Thus, at most d vertices, namely those in $V \setminus V'$, are marked. □

Finally, we show that for the parameter $|V|$ number of vertices Temporal Cluster Editing does not a admit polynomial kernel unless NP ⊆ coNP/poly.

Proposition 8.3. Temporal Cluster Editing *parameterized by the number $|V|$ of vertices does not admit a polynomial kernel unless NP ⊆ coNP/poly.*

Proof. We provide an AND-cross-composition (for a definition see Section 2.3) from classic Cluster Editing. Intuitively, we can just string together instances in the time axis such that the large instance can be transformed into a cluster graph where every vertex can be marked if and only if all of the original instances are YES-instances.

We define an equivalence relation R as follows: Two instances (G_1, k_1) and (G_2, k_2) are equivalent under R if and only if $k_1 = k_2$ and $|V(G_1)| = |V(G_2)|$. Clearly, R is a polynomial equivalence relation.

Now let $(G_1, k_1), \ldots, (G_n, k_n)$ be R-equivalent instances of Cluster Editing. Then there is an integer $k \in \mathbb{N}$ such that $k = k_i$ for every $i \in [n]$. Moreover, since the names of the vertices are not important for the problem and $|V(G_i)| = |V(G_j)|$ for every $i, j \in [n]$, we can assume without loss of generality that there is a set V such that $V = V(G_i)$ for every $i \in [n]$. Hence, $((V, E_1, E_2, \ldots, E_n), k, d)$, where $d = |V|$, is a valid instance of Temporal Cluster Editing.

This instance can be constructed in polynomial time and no extra vertices are added, hence $|V|$ is upper-bounded by the maximum size of an input instance. Furthermore, as we are allowed to mark all vertices, it follows directly from the definition of Temporal Cluster Editing that $((V, E_1, E_2, \ldots, E_n), k, d)$, is a YES-instance if and only if for every $i \in [n]$ it is possible to turn G_i into a cluster graph by at most k edge modifications.

Since CLUSTER EDITING is NP-hard [BBC04] and we AND-cross-composed it into TEMPORAL CLUSTER EDITING parameterized by $|V|$, the result follows. □

8.3 An Algorithm for Temporal Cluster Editing

In this section, we present an FPT-algorithm for TEMPORAL CLUSTER EDITING with respect to the combined parameter $k+d$. Formally, we show the following result.

Theorem 8.4. TEMPORAL CLUSTER EDITING *can be solved in* $k^{O(k+d)} \cdot |V|^3 \cdot \ell$ *time.*

To prove this theorem, we describe a recursive search-tree algorithm (see Algorithm 5) that is an extension of a simple branching algorithm to solve CLUSTER EDITING on static graphs that was first described by Gramm et al. [Gra+05] but implicitly already observed by Cai [Cai96]. For CLUSTER EDITING on static graphs the algorithm roughly works as follows: We find an induced P_3 and then try the three possibilities to destroy this P_3, namely either adding the missing edge, or removing one of the two edges in the P_3.

To use a similar strategy in the temporal case we have to perform a preprocessing step first that makes all layers look the same. More specifically, our algorithm expects some initial modification sets as input that, when applied to all layers, makes them equal up to marked vertices. These initial edge modification sets are computed greedily, hence the algorithm follows the greedy localization approach [Deh+04] in which we make decisions greedily and possibly revert them through branching later on. The greedy decisions herein give us some structure that we can exploit to keep the search-tree size bounded in k and d. The edge modification sets M_i represent both the greedy decisions and those that we made through branching. The set B contains only those made by branching. More specifically, the algorithm expects the following input:

- An instance I of TEMPORAL CLUSTER EDITING consisting of a temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers k and d.
- A *constraint* $P = (D, (M_i)_{i\in[\ell]}, B)$, consisting of a set of *marked* vertices $D \subseteq V$, edge modification sets $M_1, \ldots, M_\ell \subseteq \binom{V}{2}$, and a set $B \subseteq \binom{V\setminus D}{2}$ of *permanent* vertex pairs.

Intuitively, the constraint describes decisions that have been made by the algorithm earlier in the search-tree. Moreover, we require the constraint given to the recursive algorithm to be aligning. A constraint $P = (D, (M_i)_{i\in[\ell]}, B)$ is *aligning* if $G'_i[V \setminus D] = G'_j[V \setminus D]$ for all $i, j \in [\ell]$, where $G'_i = (V, E_i \oplus M_i)$ for all $i \in [\ell]$.

Algorithm 5 TEMPORAL CLUSTER EDITING

Input:
- A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers k and d.
- A set of marked vertices D and edge modification sets $M_1, \ldots, M_\ell$.
- A set $B \subseteq \binom{V\setminus D}{2}$ of permanent vertex pairs.

1: Apply the first applicable rule in the following ordered list:
- Rule 0
- Clean-up Rule
- Branching Rule 1
- Branching Rule 2
- Branching Rule 3

2: If none of the rules applies, then return YES.

The initial modification sets are constructed according to the following rule that adds all edges that appear in at least half of all layers to all of the remaining layers and removes all other edges.

Greedy Rule. Let $M_i = \emptyset$ for every $i \in [\ell]$. For every vertex pair $\{u, v\} \in \binom{V}{2}$ do the following:

- If $|\{E_i \mid \{u, v\} \in E_i\}| \geq \frac{\ell}{2}$, then for all $i \in [\ell]$ set $M_i \leftarrow M_i \cup (\{\{u, v\}\} \setminus E_i)$.
- If $|\{E_i \mid \{u, v\} \in E_i\}| < \frac{\ell}{2}$, then for all $i \in [\ell]$ set $M_i \leftarrow M_i \cup (\{\{u, v\}\} \cap E_i)$.

From now on, we assume that the input constraint of the algorithm contains edge modification sets produced by the Greedy Rule, together with an empty set of marked vertices and an empty set of permanent vertex pairs. We call this constraint P_{greedy}. Note that P_{greedy} is an aligning constraint.

Throughout the algorithm, we try to maintain a *good* aligning constraint which intuitively means that the constraint can be turned into a solution (if one exists).

Definition 8.2 (Good Constraint)**.** Let I be an instance of TEMPORAL CLUSTER EDITING. A constraint $P = (D, M_1, \ldots, M_\ell, B)$ is *good* for I if I is a YES-instance and there is a solution $S = (M_1^\star, \ldots, M_\ell^\star, D^\star)$ for I such that

1. $D \subseteq D^\star$,
2. there is no $\{u, v\} \in B$ such that $u \in D^\star$, and
3. for all $i \in [\ell]$ we have $M_i \cap B = M_i^\star \cap B$.

We also say that S *witnesses* that P is good.

If a constraint is not good, we call it *bad*. It is easy to see that if we face a yes-instance, then any constraint containing an empty set of marked vertices and an empty set of permanent vertex pairs is good. We call such constraints *trivial*.

Observation 8.5. *For any YES-instance* $I = (G_1, \ldots, G_\ell, d, k)$ *of* Temporal Cluster Editing*, we have that the constraint* $P = (D = \emptyset, M_1, \ldots, M_\ell, B = \emptyset)$ *is a good constraint for* I *for any sets* $M_i \subseteq \binom{V}{2}$ *with* $i \in [\ell]$.

It is obvious that the constraint P_{greedy} is trivial. Hence if the input instance of Temporal Cluster Editing for our algorithm is a YES-instance, then the initial call is with a good constraint. The algorithm is supposed to return YES if the supplied constraint is good and NO otherwise.

Our algorithm uses various different branching rules to search for a solution to a Temporal Cluster Editing input instance. Formally, branching rules are defined as follows.

Definition 8.3 (Branching Rule)**.** A *branching rule* takes as input an instance I of Temporal Cluster Editing and an aligning constraint P and returns a set of aligning constraints $P^{(1)}, \ldots, P^{(x)}$.

When a branching rule is applied, the algorithm invokes a recursive call for each constraint returned by the branching rule and returns YES if at least one of the recursive calls returns YES, otherwise, it returns NO. For that to be correct, whenever a branching rule is invoked with a good constraint, at least one of the constraints returned by the branching rule has to be a good constraint as well. Furthermore, if a branching rule is invoked with a bad constraint, none of the constraints returned by the branching rule should be good. In this case we say that a branching rule is *safe*.

Definition 8.4 (Safeness of a Branching Rule)**.** We say that a branching rule is *safe* if the following holds:

- If the branching rule is applied on an instance of Temporal Cluster Editing together with a good constraint for that instance, then at least one of the constraints returned by the branching rule is good.
- If the branching rule is applied on an instance of Temporal Cluster Editing together with a bad constraint for that instance, then none of the constraints returned by the branching rule is good.

In the following, we introduce the branching rules used by the algorithm and prove that each of them is safe (in some cases under the condition that certain other rules are not applicable). This together with Observation 8.5 will allow us to prove by induction that the algorithm eventually finds a solution for the input instance of Temporal Cluster Editing if it is a YES-instance.

The following notion and observation will be useful for the safeness proofs.

Definition 8.5. Let I be an instance of Temporal Cluster Editing and let $P = (D, M_1, \dots, M_\ell, B)$ and $P' = (D', M'_1, \dots, M'_\ell, B')$ two constraints. We say that P' *extends* P if $D' \supseteq D$, $B' \supseteq B$, and for each $i \in [\ell]$ we have $M'_i \cap B = M_i \cap B$.

Observation 8.6. *Let I be an instance of* Temporal Cluster Editing, *let P and P' be two constraints such that P' extends P, and let S be a solution witnessing that P' is good, then S also witnesses that P is good.*

We start with a rule that checks obvious constraints and aborts the recursion if they are not fulfilled.

Rule 0. If $|D| > d$ or there is an $i \in [\ell]$ such that $|M_i \cap B| > k$, then abort the current branch and return NO.

The correctness of this rule is obvious. With the next rule we edit the subgraphs induced by all non-marked vertices into cluster graphs. Similar to classic Cluster Editing, we branch on all edits that destroy induced P_3s. Additionally, we have to take into account that it may be necessary to mark vertices because otherwise they may force us to edit too many edges in some layer.

Branching Rule 1. If there is an induced $P_3 = (\{u, v\}, \{v, w\})$ in $G'_i[V \setminus D]$ for some $i \in [\ell]$, where $G'_i = (V, E_i \oplus M_i)$, then return the following up to six constraints:

1. If $\{u, v\} \notin B$: for all $i \in [\ell]$ put $M_i^{(1)} = M_i \oplus \{\{u, v\}\}$, $D^{(1)} = D$, and $B^{(1)} = B \cup \{\{u, v\}\}$.
2. If $\{v, w\} \notin B$: for all $i \in [\ell]$ put $M_i^{(2)} = M_i \oplus \{\{v, w\}\}$, $D^{(2)} = D$, and $B^{(2)} = B \cup \{\{v, w\}\}$.
3. If $\{u, w\} \notin B$: for all $i \in [\ell]$ put $M_i^{(3)} = M_i \oplus \{\{u, w\}\}$, $D^{(3)} = D$, and $B^{(3)} = B \cup \{\{u, w\}\}$.
4. For each $x \in \{u, v, w\}$: If there is no $y \in V \setminus D$ such that $\{x, y\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{x\}$, the rest stays the same.

If none of the above possibilities applies, then return NO.[19]

[19] This technically does not fit the definition of a branching rule but we can achieve the same effect by returning trivially unsatisfiable constraints such as a constraint with $|D^{(\cdot)}| > d$ which is rejected by Rule 0.

Lemma 8.7. *Branching Rule 1 is a safe branching rule.*

Proof. It is easy to check that Branching Rule 1 is indeed a branching rule since it always modifies the pairs in the edge modifications sets of all layers, hence if the input constraint is aligning so are all output constraints. Since each output constraint extends the input constraint, by Observation 8.6, if any of the output constraints is good, then so is the input constraint.

Now we show that if the input constraint is good, at least one output constraint is. Let the input constraint $P = (D, M_1, \ldots, M_\ell, B)$ be good and let $S = (M_1^\star, \ldots, M_\ell^\star, D^\star)$ be a solution for the input instance witnessing that P is good. Since each $M_i^\star$ is a cluster editing set for G_i, it holds that, for all $i \in [\ell]$, graph $G_i^\star[V \setminus D^\star]$ does not contain a P_3 as an induced subgraph, where $G_i^\star = (V, E_i \oplus M_i^\star)$. Hence, if there is some $i \in [\ell]$ and three vertices u, v, w that induce a P_3 in $G_i'[V \setminus D]$, where $G_i' = (V, E_i \oplus M_i)$, then there are two cases.

In the first case, one of u, v, w is also in $D^\star$, say $v \in D^\star$. Note that, then, v cannot be part of any permanent vertex pair, by the definition of good constraints. Thus, the constraint that puts $v \in D$ output in the fourth part of Branching Rule 1 is good.

The second case is that $u, v, w \in V \setminus D^\star$. Then, since $G_i^\star[V \setminus D^\star]$ is a cluster graph, at least one of the vertex pairs formable from u, v, w is modified by S, that is, in $M_i^\star$. Say $\{u, v\} \in M_i^\star$. Since the solution is consistent, $\{u, v\}$ either appears in $G_i^\star$ for all $i \in [\ell]$ or in none of them. Note that $\{u, v\}$ cannot be permanent since otherwise we already have that $\{u, v\} \in M_i$ by the definition of a good constraint. Thus the constraint which adds $\{u, v\}$ to M_i and makes it permanent is good. Hence, the rule is safe. □

The next rule keeps the sets of edge modifications M_i free of marked vertices. Pairs in M_i can become marked if vertices of vertex pairs processed by the Greedy Rule are marked by other branching rules further down the search tree. We invoke this rule at the beginning of each recursive call to modify the constraint before the applicability of other rules is tested.

Clean-up Rule. For each $i \in [\ell]$ and each $\{u, v\} \in M_i$: If $\{u, v\} \cap D \neq \emptyset$, then remove $\{u, v\}$ from M_i.

To show the safeness of this rule, we can formally treat the Clean-up Rule as a special case of a branching rule, that is, it produces one constraint.

Lemma 8.8. *The Clean-up Rule (when treated as a branching rule) is a safe branching rule.*

Proof. It is easy to check that Clean-up Rule is indeed a branching rule since it only removes vertex pairs that contain marked vertices from the edge modification sets, hence if the input constraint is aligning so are all output constraints. Note that permanent vertex pairs cannot contain marked vertices by the definition of constraints. It follows that the Clean-up Rule does not add or remove permanent vertex pairs from any set M_i. Furthermore, it does not change the sets D and B. It follows that the input constraint cannot become bad if it was good or vice versa. Hence, the Clean-up Rule is safe. □

The next rule tries to repair any budget violations that might occur. Since with the Greedy Rule we greedily make decisions in the beginning we expect that some of the choices were not correct. This rule will then revert these choices. Also, to have a correct estimate of the sizes of the current edge modification sets, this rule requires that the Clean-up Rule was applied.

Branching Rule 2. If there is an M_i for some $i \in [\ell]$ with $|M_i| > k$, then take any set $M'_i \subseteq M_i \setminus B$ such that $|M'_i| + |B \cap M_i| = k+1$ and return the following constraints:

1. For each $\{u,v\} \in M'_i$ return a constraint in which for all $j \in [\ell]$ we put $M_j^{(\cdot)} = M_j \oplus \{\{u,v\}\}$, $D^{(\cdot)} = D$, and $B^{(\cdot)} = B \cup \{\{u,v\}\}$.

2. For each $\{u,v\} \in M'_i$:
   - If there is no $x \in V \setminus D$ such that $\{u,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, $B^{(\cdot)} = B$, and for all $j \in [\ell]$ we put $M_j^{(\cdot)} = M_j \setminus \{\{u,v\}\}$.
   - If there is no $x \in V \setminus D$ such that $\{v,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, $B^{(\cdot)} = B$, and for all $j \in [\ell]$ we put $M_j^{(\cdot)} = M_j \setminus \{\{u,v\}\}$.

Lemma 8.9. *If the Clean-up Rule was applied and Rule 0 is not applicable, then Branching Rule 2 is a safe branching rule.*

Proof. It is easy to check that Branching Rule 2 is indeed a branching rule since it always modifies the pairs in the edge modifications sets of all layers, hence if the input constraint is aligning, so are all output constraints. Since each output constraint extends the input constraint, by Observation 8.6, if any of the output constraints is good, then so is the input constraint.

Now we show that if the input constraint is good, at least one output constraint is. Let $P = (D, M_1, \ldots, M_\ell, B)$ be the input constraint. Suppose that P is good and let $S = (M_1^\star, \ldots, M_\ell^\star, D^\star)$ be a solution for the input instance witnessing that P is good. Since Rule 0 is not applicable, we have $|M_i \cap B| \leq k$ and, thus, $M_i \setminus B \neq \emptyset$.

Since $|M_i'| + |M_i \cap B| = k+1$, $M_i \cap B \subseteq M_i^\star$, and $|M_i^\star| \leq k$, we have $M_i' \setminus M_i^\star \neq \emptyset$, i.e., there is at least one vertex pair $\{u,v\} \in M_i'$ such that $\{u,v\} \notin M_i^\star$. The branching rule creates constraints for each possible vertex pair in M_i' to remove it from M_i. Thus, in particular, there is one output constraint where $\{u,v\}$ is removed from M_i.

If $\{u,v\} \cap D^\star = \emptyset$, then, since the solution is consistent, either $\{u,v\} \in E_i \oplus M_i^\star$ for all $i \in [\ell]$ or $\{u,v\} \notin E_i \oplus M_i^\star$ for all $i \in [\ell]$. However, since P is aligning, we also have that $\{u,v\} \in E_i \oplus M_i$ for all $i \in [\ell]$ or $\{u,v\} \notin E_i \oplus M_i$ for all $i \in [\ell]$ and furthermore, $\{u,v\} \in E_i \oplus M_i$ if and only if $\{u,v\} \notin E_i \oplus M_i^\star$. Since we have that $\{u,v\} \in E_i \oplus M_i$ if and only if $\{u,v\} \notin E_i \oplus M_i \oplus \{\{u,v\}\}$, one of the constraints in the first case is good.

Otherwise at least one endpoint of $\{u,v\}$ is marked in S implying that one of the constraints in the second case is good. □

The last rule, Branching Rule 3, requires that all other rules are not applicable. In this case the non-marked vertices induce the same cluster graph in every layer. Branching Rule 3 checks whether in every layer it is possible to turn the whole layer (including the marked vertices) into a cluster graph such that the cluster graph induced by the non-marked vertices stays the same and the edge modification budget is not violated in any layer. If this is not the case for a layer i, then we will see that there are essentially two reasons for that. Either, (a), a modification in M_i that was added greedily introduced many P_3's containing marked vertices and the only way to remedy it is to roll back this modification. Or, (b), in order to make layer i a cluster graph including the marked vertices, we need to mark more vertices or make more edits outside of the marked vertices. Both cases will be treated by Branching Rule 3 simultaneously. Since M_i has bounded size, branching on the possibilities to roll back one of the edits (Case (a)) already results in a bounded number of branches. These possibilities are tested in Step 1 of Branching Rule 3. Case (b) is treated in Steps 2, 3, and 4. However, we need additional processing to upper-bound the number of vertex markings or edge edits that we need to consider. To obtain the bound, we introduce a modified version of a known kernelization algorithm [Gra+05] for classic Cluster Editing. We call this algorithm $\mathscr{K}$ and it takes as input a tuple (G, s, D, O). Herein, G will represent the current, modified state of a layer, D the currently marked vertices, s the number of edits still allowed, and O a set of vertex pairs that are *obligatory*, meaning that they cannot be modified anymore. Algorithm $\mathscr{K}$ either outputs a distinct failure symbol or two sets R and C, where R contains all unmarked vertex pairs modified by $\mathscr{K}$ and C contains all unmarked vertex pairs of the produced kernel which are not obligatory. (A vertex pair is unmarked if it does not contain a marked vertex.) In the following we give a formal description.

Modified Kernelization Algorithm $\mathscr{K}$. Given an input (G, s, D, O). First, set all vertex pairs in O to *obligatory* and exhaustively apply the following modified versions of standard data reduction rules for CLUSTER EDITING. Let $R = \emptyset$. Then, apply the following rules until none applies anymore.

K1. If $s < 0$ or there is an induced P_3 where all vertex pairs are obligatory, then abort and output a failure symbol.

K2. If a vertex pair $\{u, v\}$ is contained in the vertex set of $s+1$ distinct induced P_3s of G, then, if $\{u, v\}$ is obligatory, abort and output a failure symbol, otherwise modify $\{u, v\}$, set it to obligatory, and decrease s by one. If $u \notin D$ and $v \notin D$, then add $\{u, v\}$ to R.

K3. If there is an isolated clique, then remove it.

Let $G^{(R)}$ be the resulting graph. If the number of vertices in $G^{(R)}$ is larger than $s^2 + 2s$, then abort and output a failure symbol. Otherwise, let C be the set of all unmarked vertex pairs in $G^{(R)}$ which are not obligatory. Output R and C. This concludes the description of $\mathscr{K}$.

In the description of the branching rule, we use the following notation. For all $i \in [\ell]$ we use $\mathscr{M}_i$ to denote the set of all possible edge modifications that turn $G'_i = (V, E_i \oplus M_i)$ into a cluster graph, and where each edge of the modification set is incident to at least one marked vertex. More specifically, we have

$$\mathscr{M}_i = \{M \subseteq \tbinom{V}{2} \mid \ \forall e \in M : e \cap D \neq \emptyset \ \wedge \ G''_i = (V, E_i \oplus (M_i \cup M)) \text{ is a cluster graph}\}.$$

Note that, since each $G'_i \setminus D$ is a cluster graph, each set $\mathscr{M}_i$ is non-empty.

Branching Rule 3. If there is an $i \in [\ell]$ such that $\min_{M \in \mathscr{M}_i} |M| > k - |M_i|$, then let $M'_i = M_i \setminus B$ and invoke the modified kernelization algorithm $\mathscr{K}$ on $(G'_i, k - |M_i|, D, M_i \cap B)$, where $G'_i = (V, E_i \oplus M_i)$. If $\mathscr{K}$ outputs a failure symbol and $M'_i = \emptyset$, then return NO. If $M' \neq \emptyset$, then return the following constraints:

1. For each $\{u, v\} \in M'_i$:
   - If there is no $x \in V \setminus D$ such that $\{u, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, $B^{(\cdot)} = B$, and for each $j \in [\ell]$ with $M_j^{(\cdot)} = M_j \setminus \{\{u, v\}\}$.
   - If there is no $x \in V \setminus D$ such that $\{v, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, $B^{(\cdot)} = B$, and for each $j \in [\ell]$ with $M_j^{(\cdot)} = M_j \setminus \{\{u, v\}\}$.

- Return a constraint in which for all $j \in [\ell]$ we put $M_j^{(\cdot)} = M_j \oplus \{\{u,v\}\}$, $D^{(\cdot)} = D$, and $B^{(\cdot)} = B \cup \{\{u,v\}\}$.

If $\mathscr{K}$ does not output a failure symbol, then let R and C be the sets output by $\mathscr{K}$ and return the following constraints:

2. For each $\{u,v\} \in R$:
   - If $u \notin D$ and there is no $x \in V \setminus D$ such that $\{u,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, $B^{(\cdot)} = B$, and for each $j \in [\ell]$ with $M_j^{(\cdot)} = M_j \setminus \{\{u,v\}\}$.
   - If $v \notin D$ and there is no $x \in V \setminus D$ such that $\{v,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, $B^{(\cdot)} = B$, and for each $j \in [\ell]$ with $M_j^{(\cdot)} = M_j \setminus \{\{u,v\}\}$.
3. If $R \neq \emptyset$, then output a constraint with $D^{(\cdot)} = D$, $B^{(\cdot)} = B \cup M_i \cup R$, and $M_j^{(\cdot)} = M_j \oplus R$ for each $j \in [\ell]$.
4. For each $\{u,v\} \in C$:
   - If there is no $x \in V \setminus D$ such that $\{u,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, and the rest stays the same.
   - If there is no $x \in V \setminus D$ such that $\{v,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, and the rest stays the same.
   - Return a constraint with $D^{(\cdot)} = D$, $B^{(\cdot)} = B \cup \{\{u,v\}\}$, and $M_j^{(\cdot)} = M_j \oplus \{\{u,v\}\}$ for each $j \in [\ell]$.

Lemma 8.10. *If the Clean-up Rule was applied and Branching Rules 1 and 2 are not applicable, then Branching Rule 3 is a safe branching rule.*

Proof. It is easy to check that Branching Rule 3 is indeed a branching rule since it always modifies the edge modifications sets of all layers, hence if the input constraint is aligning so are all output constraints. Since each output constraint extends the input constraint, by Observation 8.6, if any of the output constraints is good, then so is the input constraint.

Now we show that if the input constraint is good, at least one output constraint is. Let the input constraint $P = (D, M_1, \ldots, M_\ell, B)$ be good and let $S = (M_1^\star, \ldots, M_\ell^\star, D^\star)$ be a solution for the input instance witnessing that P is good. For each layer i, Branching Rule 3 checks the minimum number of edge modifications involving at least one marked vertex to turn G_i' into a cluster graph. Since $G_i'[V \setminus D]$ is already a

cluster graph, this number always exists. Since Branching Rule 3 is applicable, there is a layer $i \in [\ell]$ such that $\min_{M \in \mathcal{M}_i} |M| > k - |M_i|$. Fix this layer i in the following.

Suppose that there is a vertex pair $\{u,v\} \in M_i \setminus M_i^\star$. Since P is good, we have $M_i \cap B = M_i^\star \cap B$, giving $\{u,v\} \in M_i' = M_i \setminus B$. Thus, $M_i' \neq \emptyset$ which means that the branch is not rejected after applying $\mathscr{K}$. In other words, there is one modification in M_i which is not in the solution witnessing that P is good, similar to Branching Rule 2. It follows from an analogous argumentation to the one in the proof of Lemma 8.9 that Branching Rule 3 produces a good constraint in Step 1. That is, Branching Rule 3 is safe in this case. Thus, from now on we assume $M_i \subseteq M_i^\star$.

We claim that $\mathscr{K}$ does not produce a failure symbol. In fact, we now show the stronger statement that $\mathscr{K}$ produces R and C such that $R \subseteq M_i^\star \setminus M_i$. To obtain this, we show the following Invariant (I) to hold before and after each application of a rule of $\mathscr{K}$. Invariant (I) states that

(i) $\mathscr{K}$ has not produced a failure symbol,

(ii) each edit made by $\mathscr{K}$ is in $M_i^\star$, and

(iii) $s = k - |M_i| - |L|$, where L is the set of modifications made by $\mathscr{K}$ so far.

Clearly, (I) holds in the beginning of K, before any application of a rule. Since Rule K2 is the only rule that makes modifications, and it clearly maintains (I) (iii), we will focus on (I) (i) and (ii). Furthermore, (I) is clearly maintained by Rule K3. It remains to treat Rules K1 and K2.

Consider Rule K1. Let L be the set of modifications made by $\mathscr{K}$ so far. By (I) (ii) we have $L \subseteq M_i^\star$. Observe that $L \cap M_i = \emptyset$ since each pair in M_i is obligatory. Hence, $(L \cup M_i) \subseteq M_i^\star$ which, since $M_i^\star$ is part of a solution, implies that there are no induced P_3s where all three vertex pairs are obligatory. Furthermore, we have that $|M_i^\star| \geq |M_i| + |L|$ and hence $k \geq |M_i^\star| \geq |M_i| + |L|$. By (I) (iii) we have $s = k - |M_i| - |L|$. Thus, $s \geq |M_i^\star| - |M_i| - |L| \geq 0$, meaning that no failure symbol is produced by Rule K1. Hence, Rule K1 maintains Invariant (I).

Now consider Rule K2. Assume that the pair $\{u,v\}$ edited by Rule K2 is not in $M_i^\star$. Since Rule K2 applies, there are $s+1$ distinct P_3s contained in the current graph $G^{(L)} := (V, E_i \oplus (M_i \cup L))$, where L are the modifications made by $\mathscr{K}$ so far. As P is good, $G^\star := (V, E_i \oplus M_i^\star)$ is a cluster graph. To compare $G^{(L)}$ and $G^\star$, recall that $L \uplus M_i \subseteq M_i^\star$, where $\uplus$ denotes a disjoint union: $M_i \subseteq M_i^\star$ by the considerations above, $L \subseteq M_i$ by (I) (ii), and $L \cap M_i = \emptyset$ because each pair in M_i is obligatory. Hence, for each of the induced P_3s in $G^{(L)}$ there is at least one distinct vertex pair in $M_i^\star \setminus (L \cup M_i)$. Thus, $|M_i^\star| \geq |L| + |M_i| + s + 1$. Since $s = k - |M_i| - |L|$, we have $|M_i^\star| \geq k+1$, a contradiction

to the fact that $M_i^\star$ is part of a solution. Thus, indeed $\{u,v\} \in M_i^\star$. It follows that Invariant (I) is maintained by Rule K2.

By Invariant (I), after applying all rules in $\mathscr{K}$ we have $L \subseteq M_i^\star \setminus M_i$, where $L = R$ is the set of modifications made by $\mathscr{K}$. We now bound the number of vertices in $G^{(R)}$. Since $s = k - |M_i| - |L|$ which we obtain from Invariant (I) (iii), we have $|M_i^\star \setminus (M_i \cup L)| \leq s$ (recall that $M_i \cap L = \emptyset$). Each vertex in $G^{(R)}$ is contained in an induced P_3. Each such P_3 contains a pair of $M_i^\star \setminus (M_i \cup L)$. Each such pair is contained in at most s P_3s by inapplicability of Rule K2. Thus, graph $G^{(R)}$ contains at most s^2+2s vertices. Thus, $\mathscr{K}$ does not produce a failure symbol. Furthermore, by Invariant (I) (ii), $R \subseteq M_i^\star$ and, moreover, since no modification made by $\mathscr{K}$ is in M_i, $R \subseteq M_i^\star \setminus M_i$. Thus, $\mathscr{K}$ produces the sets $R \subseteq M_i^\star \setminus M_i$ and C as required.

Suppose that for one edge modification $\{u,v\} \in R$ we have that $\{u,v\} \cap D^\star \neq \emptyset$, say $u \in D^\star$. Since P is good, property (ii) of being good gives that there is no pair $\{u,w\} \in B$ for any $w \in V$. Thus, Branching Rule 3 outputs a good constraint in Step 2. Hence, we now assume that R does not contain edge modifications containing vertices from $D^\star$.

Suppose that $R \neq \emptyset$. As argued above, $R \subseteq M_i^\star \setminus M_i$. Since $D \subseteq D^\star$ and no pair in R contains a vertex of $D^\star$, we have that the constraint produced in Step 3 is good. Thus, we now assume that $R = \emptyset$.

Suppose that C contains a pair which contains a vertex in $D^\star$, say u. By property (ii) of being good, there is no pair $\{u,w\} \in B$ for any $w \in V$. Thus, one of the first group of constraints produced in Step 4 is good. Thus, we now assume that no pair in C contains a vertex in $D^\star$ and hence also no pair in C contains a vertex in D.

Finally, we claim that $M_i^\star \cap C \neq \emptyset$. Suppose that $M_i^\star \cap C = \emptyset$. Since $R = \emptyset$, we have that $G^{(R)}$ is G_i' with some isolated cliques removed. Let $\widehat{M}_i$ be M_i restricted to $G^{(R)}$ and, similarly, $\widehat{M}_i^\star$ be $M_i^\star$ restricted to $G^{(R)}$. Note that, since $M_i \subseteq M_i^\star$ and $|M_i^\star| \leq k$, we have $|M_i^\star \setminus M_i| \leq k - |M_i|$ and, thus, also $|\widehat{M}_i^\star \setminus \widehat{M}_i| \leq k - |M_i|$. Since $(V, E_i \oplus M_i^\star)$ is a cluster graph, also its subgraph induced by $V(G^{(R)})$ is a cluster graph, and, hence, also $(V, E_i \oplus (\widehat{M}_i^\star \cup M_i))$ is a cluster graph, since the last two only differ in the isolated cliques.

Every pair of unmarked vertices in $G^{(R)}$ is in $\widehat{M}_i \cup C$ by the definition of C. Hence, $\widehat{M}_i \subseteq \widehat{M}_i^\star$ and $\widehat{M}_i^\star \cap C = \emptyset$ implies $\widehat{M}_i^\star \setminus \widehat{M}_i \subseteq \binom{V}{2} \setminus \binom{V \setminus D}{2}$, and, therefore, $(\widehat{M}_i^\star \setminus \widehat{M}_i) \in \mathscr{M}_i$. As $|\widehat{M}_i^\star \setminus \widehat{M}_i| \leq k - |M_i|$ and $(V, E_i \oplus (\widehat{M}_i^\star \cup M_i))$ is a cluster graph, this contradicts $\min_{M \in \mathscr{M}_i} |M| > k - |M_i|$. Thus, indeed $M_i^\star \cap C \neq \emptyset$. It follows that one of the last group of constraints produced in Step 4 is good. □

With Branching Rule 3 we can present the complete algorithm—see Algorithm 5. To prove correctness of the algorithm, we first argue that, whenever the algorithm

outputs YES, then the input instance of Temporal Cluster Editing was indeed a YES-instance. This follows in a straightforward manner from the fact that, if the algorithm outputs YES, then none of the branching rules is applicable.

Lemma 8.11. *Given an instance I of* Temporal Cluster Editing, *if Algorithm 5 outputs YES on input I and the constraint P_{greedy}, then I is a YES-instance.*

Proof. Let I be the input instance of Temporal Cluster Editing. If the algorithm outputs YES, then there is an aligning constraint $P = (D, M_1, \ldots, M_\ell, B)$ such that for all $e \in M_i$ we have that $e \cap D = \emptyset$, and none of the branching rules are applicable. Let $D^\star = D$ and for every $i \in [l]$ let $M_i' = \arg\min_{M \in \mathscr{M}_i} |M|$ and $M_i^\star = M_i \cup M_i'$, where $\mathscr{M}_i$ is as defined for Branching Rule 3. In the following we show that $S = (M_1^\star, \ldots, M_\ell^\star, D^\star)$ is a solution for I (witnessing that P is good).

Since Branching Rule 3 is not applicable, we know that $|M_i'| \leq k - |M_i|$ and hence $|M_i \cup M_i'| \leq k$. Also, since Rule 0 is not applicable, we know that $|D^\star| = |D| \leq d$. Let $G_i^\star = (v, E_i \oplus M_i^\star)$ for all $i \in [\ell]$. For all $i, j \in [\ell]$ we have that $G_i^\star[V \setminus D] = G_j^\star[V \setminus D]$ since the constraint P is aligning, and M_i' contains no unmarked pairs. Furthermore, for all $i \in [\ell]$ we have that $G_i^\star$ is a cluster graph by the definition of $\mathscr{M}_i$. □

It remains to show that, whenever the input instance I of the algorithm is a YES-instance, then the algorithm outputs YES. To this end, we define the *quality* of a good constraint and show that the algorithm increases the quality until it eventually finds a solution or determines that there is none.

Definition 8.6 (Quality of a constraint)**.** Let $I = (\mathscr{G} = (V, (E_i)_{i \in [\ell]}), k, d)$ be an instance of Temporal Cluster Editing. The *quality* $\gamma_I(P)$ of a constraint $P = (D, M_1, \ldots, M_\ell, B)$ for I is $\gamma_I(P) = |D| + |B|$.

Lemma 8.12. *Let P be a good constraint for a yes-instance of* Temporal Cluster Editing. *If applicable, each of Branching Rules 1, 2, and 3 returns a good constraint with strictly increased quality in comparison to P.*

Proof. We show the claim individually for each of the rules. We consider each of the possible returned constraints P' and show that, assuming that P' is good, then the quality of P' is strictly larger than P.

Consider Branching Rule 1. In the first three cases, the branching rule increases $|B|$. In the remaining cases, the branching rule increases $|D|$. Branching Rule 2 increases‘$|B|$ in the first case and $|D|$ in the second case. Branching Rule 3 also increases $|B|$ or $|D|$ in each of the four steps. □

We can now show the correctness of Algorithm 5. Lemma 8.11 ensures that we only output `true` if the input is actually a yes-instance and Lemma 8.12 together with the safeness of all branching rules ensures that if the input is a yes-instance, the algorithm outputs `true`.

Proposition 8.13 (Correctness of Algorithm 5)**.** *Given a* Temporal Cluster Editing *instance I, Algorithm 5 outputs YES on input I and the initial constraint P_{greedy} if and only if I is a YES-instance.*

Proof. By Lemma 8.11, if Algorithm 5 outputs YES on input I and the initial constraint P_{greedy}, then I is a YES-instance. It remains to show the other direction.

Let I be a YES-instance of Temporal Cluster Editing. By Observation 8.5 we have that P_{greedy} is a good constraint. Note that the order in which rules are applied (see Algorithm 5) ensures safeness for all branching rules (Lemmata 8.7, 8.8, 8.9, and 8.10). Furthermore, by Lemma 8.12 we have that all branching rules except the Clean-up Rule strictly increase the quality of a good constraint. It is easy to see that the Clean-up Rule does not decrease the quality of a good constraint and it is applied only once before either one of the other rules apply or the algorithm terminates. Let $P_{\max}$ be a good constraint with the highest quality produced during the run of the algorithm. Since the quality is an integer bounded from above by $|V|+\binom{|V|}{2}$, there must be such a constraint. As the Clean-up Rule does not decrease the quality, we can also assume that it was exhaustively applied to $P_{\max}$. If any of the branching rules would apply to $P_{\max}$, then, by safeness of the branching rules and Lemma 8.12, it would produce a good constraint of strictly higher quality, contradicting the maximality of $P_{\max}$. Hence the algorithm run on $P_{\max}$ returned `true` and, therefore, the whole algorithm returned YES. □

It remains to show that Algorithm 5 has the claimed running time upper bound. We can check that all branching rules create at most $O(k^4)$ recursive calls. The preprocessing by the Greedy Rule and the alignment of the constraints ensures that the edge modification sets in sufficiently many layers increase for the search tree to have depth of at most $O(k+d)$. The time needed to apply a branching rule is dominated by Branching Rule 3, where we essentially have to solve classic Cluster Editing in every layer.

Proposition 8.14. *The running time of Algorithm 5 is $k^{O(k+d)} \cdot |V|^3 \cdot \ell$.*

Proof. We bound the running time of Algorithm 5 by the following straightforward approach. Note that the recursive calls of Algorithm 5 define a tree in which each node corresponds to a call of Algorithm 5 and two nodes are connected by an edge

if one of the corresponding calls of the algorithm is a recursive call of the other. The tree is rooted at the node corresponding to the initial constraint P_{greedy}. Note that P_{greedy} can be computed in $O(|V|^2 \cdot \ell)$ time. We first bound the size of the search tree, and then the computation spent in each node of the search tree. Note that we apply Clean-up Rule at the beginning of each recursive call, that is, without creating further recursive calls. Hence we call this rule *degenerate*.

To bound the depth of the search tree, the length of a path from the root to the farthest leaf, we show that each (nondegenerate) branching rule increases either $|D|$ by at least one or it increases $\sum_{1 \leq i \leq \ell} |M_i \cap B|$ by at least $\frac{\ell}{2}$. If $|D| > d$ or $\sum_{1 \leq i \leq \ell} |M_i \cap B| > \ell \cdot k$, then the algorithm terminates (Rule 0). Before we show this, we prove an invariant that for every constraint produced by the algorithm and for every $\{u, v\} \in M_i \setminus B$ for some i with $u, v \notin D$ we have $|\{j \mid \{u, v\} \in M_j\}| \leq \frac{\ell}{2}$. To this end, note that is initially fulfilled when P_{greedy} is computed by the Greedy Rule and whenever any other rule touches a pair $\{u, v\}$ then in the produced constraint we have that $\{u, v\} \in B$, $u \in D$, or $v \in D$. That is, these rules cannot break the invariant.

Consider Branching Rule 1. In the first three cases $\sum_{1 \leq i \leq \ell} |M_i \cap B|$ increases by at least $\frac{\ell}{2}$, since the pair was not in B before the application of the rule and thus could appear in M_i for at most $\frac{\ell}{2}$ different layers i by the above proven invariant and, hence, after the application of the rule it appears in $|M_i \cup B|$ for at least $\frac{\ell}{2}$ different layers i. By a similar argument, also Branching Rules 2 and 3 increase either $|D|$ by one or $\sum_{1 \leq i \leq \ell} |M_i \cap B|$ by at least $\frac{\ell}{2}$. Hence, we can upper-bound the depth of the search tree with $2k + d$.

Observe that the number of children of each node in the search tree where Branching Rule 1 or 2 is applied is upper-bounded by $3k + 3$. Branching Rule 3 creates at most $3k$ recursive calls in the first step. In the second step it creates at most $2|R|$ recursive calls, at most one in the third step and in the fourth step the number of recursive calls created is at most $3|C|$. By the description of the modified kernelization algorithm $\mathscr{K}$ we have $|R| \leq s$ and $|C| \leq (s^2 + 2s)^2$. By the way $\mathscr{K}$ is invoked by Branching Rule 3 we have $s \leq k$ and, thus, the number of recursive calls is $O(k^4)$. It follows that the size of the whole search tree is in $k^{O(k+d)}$.

The Clean-up Rule plays a special role. The Clean-up Rule can be exhaustively applied in $O(|\binom{V}{2}| \cdot \ell) = O(|V|^2 \cdot \ell)$ time on the beginning of each recursive call.

Lastly, we analyze for each rule, how much time is needed to check whether the rule is applicable and, if so, to compute the constraints it outputs. To check the applicability of Branching Rule 1, the algorithm needs to check whether there is a layer containing an induced P_3. This can be done in $O(|V| + m)$ time, where m is the maximum number of edges in a layer. Hence, overall we need $O((|V| + m) \cdot \ell)$ time to check whether Branching Rule 1 is applicable and in this time we can also

compute the output constraints. In the case of Branching Rule 2, we need $O(|V|^2 \cdot \ell)$ time to check whether it is applicable and to output the constraints. For the last rule, Branching Rule 3, we need to check whether the set families $\mathcal{M}_i$ are nonempty. To do this we essentially need to solve CLUSTER EDITING on each layer to check whether the rule is applicable. This can be done in $O(3^k \cdot (|V| + m))$ time, similarly to the straightforward algorithm for CLUSTER EDITING. That is, recursively find a P_3 and branch into the at most three cases of modifying vertex pairs in the P_3 which contain at least one marked vertex. The time to compute the constraints is dominated by the application of the modified kernelization algorithm $\mathscr{K}$. The data reduction rules of $\mathscr{K}$ can be applied exhaustively in $O(|V|^3)$ time [Gra+05]. Hence, overall, the algorithm has running time $k^{O(k+d)} \cdot |V|^3 \cdot \ell$. □

This concludes the proof of Theorem 8.4, since it follows directly from Proposition 8.13 and Proposition 8.14. Furthermore, Proposition 8.3 implies that this FPT result presumably cannot be improved to yield a polynomial kernel since $k + d \leq |V|^3$.

We remark that it is not difficult to see that TEMPORAL CLUSTER EDITING can also be solved in $|V|^{O(|V|)} \cdot \ell$ time, which is incomparable to the running time of Algorithm 5 since k might be as large as $\Omega(|V|^2)$: First guess the marked vertices ($2^{|V|}$ possibilities). Then guess how many clusters (that is, disjoint cliques) there are in the modified graph induced by the non-marked vertices ($|V|$ possibilities), and for every non-marked vertex, guess to which cluster it belongs (at most $|V|^{|V|}$ possibilities). Now for every layer, independently guess how many additional clusters there are consisting only of marked vertices, and for every marked vertex, guess to which cluster it belongs (at most $|V|^{|V|} \cdot \ell$ possibilities). Finally check, whether such a solution can be obtained by at most k modifications per layer, which can be done in polynomial time.

8.4 Kernelization for Temporal Cluster Editing

In this section we present a polynomial kernel for TEMPORAL CLUSTER EDITING for the parameter combination $k + d + \ell$.

Theorem 8.15. TEMPORAL CLUSTER EDITING *admits a kernel of size* $O(\ell^3 \cdot (k+d)^4)$ *which can be computed in* $O(\ell \cdot |V|^3)$ *time.*

The rest of this section is devoted to the proof of this theorem. To this end, we introduce a number of so-called *data reduction rules*. A data reduction rule is a description of a procedure that takes a problem instance as input (in our case an instance of TEMPORAL CLUSTER EDITING) and outputs an instance of the same

problem. We say that a data reduction rule is *correct*, if it holds that the input instance is a YES-instance if and only if the output instance is a YES-instance.

We provide several data reduction rules that subsequently modify the instance and we assume that if a particular rule is to be applied, then the instance is reduced with respect to all previous rules, that is, all previous rules were already exhaustively applied. To keep track of the budget in the individual layers we introduce Temporal Cluster Editing with Separate Budgets which differs from Temporal Cluster Editing only in that, instead of a global upper bound k on the number of edits, we receive ℓ individual budgets k_i for all $i \in [\ell]$, and we require that $|M_i| \leq k_i$.

We first transform the input instance of Temporal Cluster Editing to an equivalent instance of Temporal Cluster Editing with Separate Budgets by letting $k_i = k$ for every $i \in [\ell]$. Then we apply all our data reduction rules to Temporal Cluster Editing with Separate Budgets. Finally, we show how to transform the resulting instance of Temporal Cluster Editing with Separate Budgets to an equivalent instance of Temporal Cluster Editing with just a small increase of the vertex set.

Throughout the presentation, let $(\mathcal{G} = (V, (E_i)_{i\in[\ell]}), k_1, \ldots, k_\ell, d)$ be the current instance and let $k = \max\{k_i \mid i \in [\ell]\}$.

The following rules represent well known data reduction rules for classic Cluster Editing [Gra+05] applied to the individual layers of the temporal graph. The first rule formalizes the obvious constraint on the solvability of the instance. We omit a proof of correctness for this rule.

Reduction Rule 1. If there is a layer $i \in [\ell]$ such that $k_i < 0$, then answer NO.[20]

Reduction Rule 2. If there is a layer $i \in [\ell]$ and an edge $\{u, v\} \in E_i$ in layer i such that G_i contains at least $k_i + 1$ different induced P_3s each of which contains the edge $\{u, v\}$, then remove $\{u, v\}$ from E_i and decrease k_i by one.

Lemma 8.16. *Reduction Rule 2 is correct.*

Proof. Let $I = (\mathcal{G} = (V, (E_i)_{i\in[\ell]}), k_1, \ldots, k_\ell, d)$ be the original instance and $\widehat{I} = (\widehat{\mathcal{G}} = (V, E_1, \ldots, \widehat{E_i}, \ldots E_\ell), k_1, \ldots, \widehat{k_i}, \ldots, k_\ell, d)$ be the instance after the application of the rule, where $\widehat{E_i} = E_i \setminus \{\{u, v\}\}$ and $\widehat{k_i} = k_i - 1$. If $\widehat{I}$ is a YES-instance, then I is also a YES-instance with the same solution as the one for $\widehat{I}$ and the pair $\{u, v\}$ added.

For the converse, assume that $S = (D, M_1, \ldots M_\ell)$ is a solution for I and let $G'_i = (V, E_i \oplus M_i)$. We claim that there is also a solution S' for $\widehat{I}$. Since the input temporal

[20] This technically does not fit the definition of a data reduction rule and should be read as an abbreviation for returning a trivial NO-instance, that is, an arbitrary but fixed NO-instance of constant size.

graphs in I and $\widehat{I}$ only differ by one edge $\{u,v\}$, suppose towards a contradiction that G'_i still contains $\{u,v\}$, meaning that $\{u,v\} \notin M_i$. By the assumptions of the rule we know that there are k_i+1 vertices $w_1,\ldots,w_{k_i+1}$ such that for each $i \in [k_i+1]$ the induced subgraph $G_i[\{u,v,w_j\}]$ is a P_3, which has to be destroyed to obtain a cluster graph. Since $\{u,v\} \in E_i \oplus M_i$, in order to destroy all P_3s, for each $j \in [k_i+1]$ we have to either add the absent edge to or delete an existing edge e (with $e \neq \{u,v\}$) from the induced subgraph $G[\{u,v,w_j\}]$. However, since for two different indices $j_1, j_2 \in [k_i+1]$ the pair $\{u,v\}$ is the only pair of vertices shared between $\{u,v,w_{j_1}\}$ and $\{u,v,w_{j_2}\}$, we have to modify at least k_i+1 edges, a contradiction to $|M_i| \leq k$, Hence G'_i does not contain $\{u,v\}$, i.e., $\{u,v\} \in M_i$ and S' obtained from S by replacing M_i with $M_i \setminus \{\{u,v\}\}$ is a solution to $\widehat{I}$. □

Reduction Rule 3. If there is a layer $i \in [\ell]$ and a pair $\{u,v\} \in V$ of vertices with $\{u,v\} \notin E_i$ (a non-edge) in layer i such that G_i contains at least k_i+1 different induced P_3s each of which involves both u and v, then add $\{u,v\}$ to E_i and decrease k_i by one.

Lemma 8.17. *Reduction Rule 3 is correct.*

Proof. The proof is almost the same as for Lemma 8.16, the obvious difference is that we assume $\widehat{E_i} = E_i \cup \{\{u,v\}\}$. Also in the second implication, supposing that M_i does not contain $\{u,v\}$ leads to a contradiction. □

As with the classic CLUSTER EDITING we can upper-bound the number of vertices involved in a P_3 in each layer. Let $R_i \subseteq V$ be the set of the vertices v that appear in some induced P_3 in G_i and let $R = \bigcup_{i=1}^{\ell} R_i$.

Reduction Rule 4. If there is a layer $i \in [\ell]$ such that $|R_i| > k_i^2 + 2k_i$, then answer NO.

Lemma 8.18. *Reduction Rule 4 is correct.*

Proof. Suppose towards a contradiction that $|R_i| > k_i^2 + 2k_i$ and $I = (\mathcal{G} = (V, (E_i)_{i\in[\ell]}), k_1,\ldots,k_\ell, d)$ is a YES-instance. Let $(M_1,\ldots M_\ell, D)$ be a solution to I and define $G'_i = (V, E_i \oplus M_i)$. For each modified edge $\{u,v\} \in M_i$ denote by R_{uv} the set of vertices w such that the induced subgraph $G_i[\{u,v,w\}]$ is a P_3. Since the instance is reduced with respect to Reduction Rules 2 and 3, for each modified edge $\{u,v\} \in M_i$ we have $|R_{uv}| \leq k_i$. Since G'_i is a cluster graph and, thus, does not contain a P_3 as an induced subgraph, we know that $R_i \subseteq \bigcup_{\{u,v\}\in M_i} (\{u,v\} \cup R_{uv})$. It follows that $|R_i| \leq k_i \cdot (2 + k_i) = k_i^2 + 2k_i$—a contradiction. □

As a major difference to CLUSTER EDITING for a single layer, we cannot simply remove the vertices that are not involved in any P_3 since we require the cluster graphs

in individual layers not to differ too much. We show that the vertices in the clusters that do not change can be freely removed.

Reduction Rule 5. If there is a subset $A \subseteq V \setminus R$ such that for each layer $i \in [\ell]$, the subset A is the vertex set of a connected component of G_i, then remove A (and the corresponding edges) from every G_i.

Lemma 8.19. *Reduction Rule 5 is correct.*

Proof. Let $I = (\mathcal{G} = (V, (E_i)_{i\in[\ell]}), k_1, \ldots, k_\ell, d)$ be the original instance and $\widehat{I} = (\widehat{\mathcal{G}} = (V \setminus A, (\widehat{E_i})_{i\in[\ell]}), k_1, \ldots, k_\ell, d)$ be the instance after the application of the rule, where for each $i \in [\ell]$, we have $\widehat{E_i} = E(G_i[V \setminus A])$. Since $A \cap R = \emptyset$, we have that A induces a complete subgraph in each layer $i \in [\ell]$. Moreover, for each layer $i \in [\ell]$, the set A is the vertex set of a connected component of graph G_i. Thus, $G_i[A]$ is a complete connected component in G_i, meaning that $E_i = \widehat{E_i} \cup \binom{A}{2}$.

Let $(M_1, \ldots, M_\ell, D)$ be a solution for I and let $\widehat{D} = D \setminus A$ and $\widehat{M_i} = M_i \cap \binom{V\setminus A}{2}$ for every $i \in [\ell]$. Then $(\widehat{M_1}, \ldots, \widehat{M_\ell}, \widehat{D})$ forms a solution to $\widehat{I}$.

Conversely, let $\widehat{S} = (\widehat{D}, \widehat{M_1}, \ldots, \widehat{M_\ell})$ be a solution for $\widehat{I}$. We claim that $\widehat{S}$ is also a solution for I. Indeed, each $G_i' = (V, E_i \oplus \widehat{M})$ is a cluster graph (note that A is the vertex set of a complete connected component in G_i), $|\widehat{M_i}| \leq k_i$ and for all $i, j \in [\ell]$ we have that $E_i \oplus \widehat{M_i} \cap \binom{V\setminus D}{2} = E_j \oplus \widehat{M_j} \cap \binom{V\setminus D}{2}$ since $\widehat{E_i} \oplus \widehat{M_i} \cap \binom{V\setminus(A\cup D)}{2} = \widehat{E_j} \oplus \widehat{M_j} \cap \binom{V\setminus(A\cup D)}{2}$. □

The next rule allows us to reduce vertices that appear in exactly the same clusters, if there are many.

Reduction Rule 6. If there is a set $A \subseteq V \setminus R$ with $|A| \geq k + d + 3$ such that for every layer $i \in [\ell]$ it holds that all vertices of A are in the same connected component of G_i, then select an arbitrary $v \in A$ and remove v from every G_i.

For the correctness proof of this and subsequent data reduction rules we find the following observation handy.

Observation 8.20. *Let G be a complete graph on at least $k+2$ vertices and let H be a cluster graph such that $V(G) = V(H)$. If H is not complete, then G and H differ in at least $k+1$ edges, that is, $|E(G) \oplus E(H)| \geq k+1$.*

Proof. The statement is obviously true for $k \leq 0$, let us assume that $k > 0$. Since H is a cluster graph which is not a complete graph, it must have several connected components. Let X be the smallest of these connected components and define $Y = V(G) \setminus X$. The set $E(G) \oplus E(H)$ must contain at least all the edges between X and Y, hence $|E(G) \oplus E(H)| \geq |X| \cdot |Y|$.

If $|X| \geq \frac{k+2}{2}$, then also $|Y| \geq \frac{k+2}{2}$ since X is the smallest component. But then $|X| \cdot |Y| \geq (\frac{k+2}{2})^2 = \frac{k^2+4k+4}{4} \geq k+1$.

If $|X| < \frac{k+2}{2}$, then let us denote $x = |X|$ and we have $|Y| \geq k+2-x$. We know that $|X| \cdot |Y| \geq x \cdot (k+2-x)$. The function $f(x) = x \cdot (k+2-x)$ is increasing for $x < \frac{k+2}{2}$ with $f(1) = k+1$, hence $|X| \cdot |Y| \geq k+1$, finishing the proof. □

Lemma 8.21. *Reduction Rule 6 is correct.*

Proof. Let $I = (\mathcal{G} = (V, (E_i)_{i\in[\ell]}), k_1, \ldots, k_\ell, d)$ be the original instance and $\widehat{I} = (\widehat{\mathcal{G}} = (V \setminus \{v\}, (\widehat{E_i})_{i\in[\ell]}), k_1, \ldots, k_\ell, d)$, where $\widehat{E_i} = E(G_i[V \setminus \{v\}])$ be the instance after the application of the rule. Let $(M_1, \ldots, M_\ell, D)$ be a solution to I, and let $\widehat{D} = D \setminus \{v\}$ and for every $i \in [\ell]$ let $\widehat{M_i} = M_i \cap \binom{V\setminus\{v\}}{2}$. Then $(\widehat{M_1}, \ldots, \widehat{M_\ell}, \widehat{D})$ forms a solution to $\widehat{I}$.

Conversely, let $\widehat{S} = (\widehat{D}, \widehat{M_1}, \ldots, \widehat{M_\ell})$ be a solution to $\widehat{I}$. Let w be an arbitrary vertex of $A \setminus (\widehat{D} \cup \{v\})$ (note that since $|A| \geq k+d+3$, $|\widehat{D}| \leq d$ and $k \geq 0$, the set $A \setminus (\widehat{D} \cup \{v\})$ is not empty). We will construct a solution for I such that after applying the solution we have that v is a true twin of w in every layer, that is, we will put v into the same clusters as w. Formally, for each layer $i \in [\ell]$, we define $\widehat{E_i}' = E(\widehat{G_i}) \oplus \widehat{M_i}$, $E_i' = \widehat{E_i}' \cup \{\{x, v\} \mid \{x, w\} \in \widehat{E_i}'\} \cup \{\{v, w\}\}$ and $M_i = E_i \oplus E_i'$. We claim that $(M_1, \ldots, M_\ell, \widehat{D})$ is a solution to I.

First, each $G_i' = (V, E_i')$ is a cluster graph. If there are two layers $i, j \in [\ell]$ such that $G_i' \setminus D \neq G_j' \setminus D$, then without loss of generality we can assume that there is some $x \in V \setminus (D \cup \{v\})$ such that $\{v, x\} \in E_i'$ but $\{v, x\} \notin E_j'$. But then $\{w, x\} \in \widehat{E_i}'$ and $\{w, x\} \notin \widehat{E_j}'$, a contradiction since neither w nor x is in $\widehat{D}$.

Finally, let us show that for each layer $i \in [\ell]$ we have that $|M_i| \leq |\widehat{M_i}| \leq k_i$. To this end, we first observe that all vertices of $A \setminus \{v\}$ are in the same component of $(V \setminus \{v\}, \widehat{E_i}')$. Since $A \cap R = \emptyset$ and all vertices of A are in the same connected component of G_i, we have that $\widehat{G_i}[A \setminus \{v\}]$ is complete. Hence, if $A \setminus \{v\}$ does not induce a complete subgraph in $(V \setminus \{v\}, \widehat{E_i}')$, then by Observation 8.20 we can conclude that $\widehat{M_i}$ contains at least $k+1$ edges, as $|A \setminus \{v\}| \geq k+d+2 \geq k+2$.

Now for every $x \in V \setminus A$ and every $i \in [\ell]$ we have that $\{v, x\} \in E(G_i)$ if and only if $\{u, x\} \in E(G_i)$ for every $u \in A$ as otherwise the induced subgraph $G_i[\{v, u, x\}]$ would be a P_3, contradicting $A \cap R = \emptyset$. Similarly, for every $x \in V \setminus A$ and every $i \in [\ell]$ we have that $\{v, x\} \in E_i'$ if and only if $\{u, x\} \in E_i'$ for every $u \in A$, since $(V \setminus \{v\}, \widehat{E_i}')$ is a cluster graph and since E_i' is constructed in this way. It follows that if $\{x, v\} \in M_i$ for some $x \in V \setminus A$, then $\{x, u\} \in \widehat{M_i}$ for every $u \in A \setminus \{v\}$ and $|\widehat{M_i}| \geq k+1 \geq k_i+1$—a contradiction. Hence $\{x, v\} \notin M_i$ for every $x \in V \setminus \{v\}$ and thus $M_i \subseteq \widehat{M_i}$. □

The next rule shows that the remaining clusters in a YES-instance cannot be large.

Reduction Rule 7. If there is a layer $i \in [\ell]$ and a connected component A of G_i with $|A \setminus R| \geq k + 2d + 3$, then answer NO.

Before we prove correctness of this rule, let us first make a folklore observation.

Observation 8.22. *If a connected component C of a graph has at least three vertices and is not complete, then every vertex of C appears in some induced P_3.*

Proof. Consider an arbitrary vertex $u \in V(C)$. If u is adjacent to v for every $v \in V(C) \setminus \{u\}$, then there must be some pair $\{x, y\} \subseteq V(C) \setminus \{u\}$ of vertices such that $\{x, y\} \notin E(C)$ since the component is not complete. Then, $C[\{u, x, y\}]$ is a P_3.

Otherwise u is not adjacent to some vertex $v \in V(C) \setminus \{u\}$. Then let P be the shortest path between u and v. This path has at least three vertices and each three consecutive vertices of this path induce a subgraph which is a P_3. □

Lemma 8.23. *Reduction Rule 7 is correct.*

Proof. Suppose towards a contradiction that A is a connected component of G_i for some layer $i \in [\ell]$ with $|A \setminus R| \geq k + 2d + 3$, and $(\mathcal{G} = (V, (E_i)_{i \in [\ell]}), k_1, \ldots, k_\ell, d)$ is a YES-instance. Let $(M_1, \ldots M_\ell, D)$ be a solution to the instance. For every layer $j \in [\ell]$, let $E'_j = E_j \oplus M_j$ and let $G'_j = (V, E'_j)$. Let $A' = A \setminus (D \cup R)$ and note that $|A'| \geq k + d + 3$. We claim that for every $j \in [\ell]$, all vertices of A' are in the same connected component of G_j, contradicting the instance being reduced with respect to Reduction Rule 6.

Since $A' \cap R = \emptyset$ and all vertices of A are in the same connected component of G_i, by Observation 8.22 we have that $G_i[A']$ is complete. Hence, if $G'_i[A']$ is not complete, then by Observation 8.20 we have that M_i contains at least $k + 1 \geq k_i + 1$ edges, as $|A'| \geq k + 2$, which is a contradiction. Hence, $G'_i[A']$ is complete. For every $j \in [\ell]$, since $G'_j[V \setminus D] = G'_i[V \setminus D]$, the graph $G'_j[A']$ is complete. Now again, if $G_j[A']$ is not complete for some layer $j \in [\ell]$, then again by Observation 8.20 we have that M_j contains at least $k + 1 \geq k_j + 1$ edges—a contradiction. □

Now we introduce our final rule bounding the number of vertices in the instance.

Reduction Rule 8. If $|V| > \ell \cdot (k^2 + 2k + d \cdot (k + 2d + 2) + 2k)$, then answer NO.

Lemma 8.24. *Reduction Rule 8 is correct.*

Proof. Suppose towards a contradiction that $|V| > \ell \cdot (k^2 + 2k + d \cdot (k + 2d + 2) + 2k)$ and $(\mathcal{G} = (V, (E_i)_{i \in [\ell]}), k_1, \ldots, k_\ell, d)$ is a YES-instance. Let $(M_1, \ldots M_\ell, D)$ be a solution to the instance. For each $i \in [\ell]$, let $E'_i = E_i \oplus M_i$ and let $G'_i = (V, E'_i)$. Let us denote by

$S = \bigcup_{i=1}^{\ell} \bigcup_{\{u,v\} \in M_i} \{u, v\}$ the set of vertices adjacent to any modification. Obviously, we have that $|S| \leq \ell \cdot 2k$.

For every layer $i \in [\ell]$ and every $x \in D$ let us denote by $Q_x^i \subseteq V \setminus R$ the set of vertices from $V \setminus R$ in the same connected component of G_i' as the vertex x and $Q = \bigcup_{i=1}^{\ell} \bigcup_{x \in D} Q_x^i$. Since the instance is reduced with respect to Reduction Rule 7, we know that $|Q_x^i| \leq k + 2d + 2$ and, thus, we have that $|Q| \leq \ell \cdot d \cdot (k + 2d + 2)$.

Note also that $|R| \leq \ell \cdot (k^2 + 2k)$, since the instance is reduced with respect to Reduction Rule 4. Now since $|V| > \ell \cdot (k^2 + 2k + d \cdot (k + 2d + 2) + 2k)$, $|R| \leq \ell \cdot (k^2 + 2k)$, $|Q| \leq \ell \cdot d \cdot (k + 2d + 2)$, and $|S| \leq \ell \cdot 2k$, the set $V' = V \setminus (Q \cup R \cup S)$ is not empty. Let u be an arbitrary vertex from V'. Since the instance is reduced with respect to Reduction Rule 5, we know that there are two distinct layers $i, j \in [\ell]$ and a vertex v such that u and v are in the same connected component of G_i and in different connected components of G_j. Since v is not in S, we know that the same holds for the graphs G_i' and G_j'. However, since v is neither in Q nor in R, we have that neither u nor v is in D. But then $G_i'[V \setminus D]$ and $G_j'[V \setminus D]$ are different—a contradiction. □

After bounding the size of the instance through Reduction Rule 8 it remains to transform the resulting instance of Temporal Cluster Editing with Separate Budgets to an equivalent instance of Temporal Cluster Editing. To this end we introduce new vertex set A of size exactly $2k + 2$ to V and to each E_i introduce all edges from $\binom{A}{2}$. Then, for each $i \in [\ell]$ we remove $k - k_i$ arbitrary edges between vertices of A from E_i and set $k_i = k$.

If $\{u, v\}$ is an edge removed in this step, then u and v had $2k$ common neighbors in A and by at most $k - 1$ other edge removals they could loose at most $k - 1$ of them. Hence, Reduction Rule 3 would apply to each pair of vertices from A with an edge removed. Applying Reduction Rule 3 exhaustively and then Reduction Rule 5 would revert all the changes made. Hence, the constructed instance is equivalent to the one obtained after exhaustive application of all the data reduction rules.

The constructed instance can be turned into an equivalent instance of Temporal Cluster Editing in an obvious way. Since no rule increases k, d, or ℓ, and we have that $|V| = O(\ell \cdot (k + d)^2)$, the resulting instance can be described using $O(\ell^3 \cdot (k + d)^4)$ bits and it is equivalent to the original instance, it remains to show that the kernel can be computed in polynomial time.

Lemma 8.25. *All data reduction rules introduced in this section can be exhaustively applied in $O(\ell \cdot |V|^3)$ time.*

Proof. If $|V| < k^2$, then we can output the original instance as the kernel. Let us assume that $k^2 \leq |V|$.

We can check whether Reduction Rule 1 applies in $O(\ell)$ time on the beginning and in constant time whenever any later rule changes the budget. Applying the rule takes constant time.

For each layer $i \in [\ell]$ we can count in $O(|V|^3)$ time in how many induced subgraphs isomorphic to P_3 each pair of vertices appears and classify the pairs according to that count. Then we apply Reduction Rules 2 and 3 to the pairs which appear in many P_3's. Each application takes $O(|V|)$ time and at the same time we can update the counts for affected pairs. Hence, these data reduction rules can be exhaustively applied to one layer in $O(|V|^3)$ time. Also in the same time we can determine the sets R_i and eventually apply Reduction Rule 4. Since the later rules only delete vertices or answer NO, no application of a later rule can create an opportunity to apply Reduction Rule 2, 3, or 4. Hence, these data reduction rules can be exhaustively applied to the instance in $O(\ell \cdot |V|^3)$ time.

In $O(\ell \cdot |V|^2)$ time we can compute the graphs $G_\cap = (V, \bigcap_{i=1}^{\ell} E_i)$ and $G_\cup = (V, \bigcup_{i=1}^{\ell} E_i)$. Then Reduction Rule 5 applies to all connected components of $G_\cup$ not containing vertices of R that are also connected components of $G_\cap$. All of these applications can be recognized in $O(|V|^2)$ time and all of them together applied in $O(\ell \cdot |V|^2)$ time. No application of a later rule can create an opportunity to apply Reduction Rule 5.

Reduction Rule 6 applies to each connected component of $G_\cap$ which has the appropriate number of vertices that are not in R. All of these applications can be recognized in $O(|V|^2)$ time and all of them together applied in $O(\ell \cdot |V|^2)$ time. Since later rules only answer NO, no application of a later rule can create an opportunity to apply Reduction Rule 6.

We can check whether the rule applies in $O(\ell \cdot |V|^2)$ time for Reduction Rule 7 and in constant time for Reduction Rule 8 and apply any of them in constant time.

Hence the data reduction rules can be exhaustively applied in $O(\ell \cdot |V|^3)$ time, the final reduction back to Temporal Cluster Editing takes $O(k^2) \subseteq O(|V|)$ time and the result follows. □

Theorem 8.15 is now directly implied by the correctness of all introduced data reduction rules and Lemma 8.25.

8.5 Conclusion

Our results highlight that Temporal Cluster Editing is much richer in structure than classic Cluster Editing. Techniques for the classic problem somewhat carry over but incorporating new methods, such as the greedy localization step, seem necessary. We employ the latter in our fixed-parameter algorithm for Temporal Cluster Editing with respect to the combination of k and d. However, we believe

that both the running time of the FPT-algorithm and the size of the polynomial kernel leave room for improvement. Also, lower bounds on running time and kernel size for this problem would further help to understand its computational complexity.

There are also a number of natural generalizations and restrictions for the problem that are worth considering. It might be especially interesting to put further constraints on the marking of vertices, like giving marked vertices a weight corresponding to the number of different clusters they are part of in different layers. Furthermore, canonical restrictions such as allowing only edge removals [Gra+05, SST04] might be promising candidates for future research.

An interesting generalization would be to only require that each cluster is a Δ-clique that is present over the whole lifetime of the temporal graph.

CHAPTER 9

Conclusion

In this thesis, we have seen six classic graph problems viewed from a temporal perspective. There are usually various canonical ways to transfer a static graph problem into the temporal setting. For some of the problems, we used already existing temporal models, for others we have proposed new models. An immediate observation is that polynomial-time solvable problems tend to become NP-hard when transferred to the temporal setting, even on temporal graphs with a constant lifetime. In this thesis, we have seen four examples where this is the case: RESTLESS TEMPORAL (s,z)-PATH, (RESTLESS) TEMPORAL (s,z)-SEPARATION, TEMPORAL MATCHING, and (SLIDING WINDOW) TEMPORAL COLORING (for two colors). In the following, we review the main contributions and try to put them into a larger context. We further discuss general directions for future research in the field of temporal graph algorithms. For discussions of specific open questions related to the problems we investigated, we refer to the concluding sections of the respective chapters.

9.1 Main Contributions and General Messages

In the following, we adopt a bird's-eye view on the results of this thesis and try to point out interesting similarities between the results for the problems we investigated.

Sparsity does not help. Or, as this should maybe rather say, "we have not found a good way to measure sparsity of temporal graphs yet". We say that parameters measure sparsity roughly if they have the property that if they are small, it implies that a graph has a small number of edges in comparison to the maximum number of edges that it could have. In the computational hardness results we presented in this thesis, we put an emphasis on restricting the instances produced by the reductions as much as possible, often with the goal to create sparse temporal graphs. This usually strengthens the results since it is often easy to see that computational hardness on sparse instances also implies computational hardness on non-sparse or dense instances. Let us review the results for some of the problems we considered.

- RESTLESS TEMPORAL (s,z)-PATH (Chapter 3): Here, we have shown that the problem is NP-hard on graphs with lifetime three, even if the underlying graph

has maximum degree six (Theorem 3.3). We have further shown that the problem is W[1]-hard when parameterized by the feedback vertex number of the underlying graph (Theorem 3.5). Additionally, we have that the instances produced by the reductions that we used to show these results have the property, that every edge appears in at most one layer.

- (Restless) Temporal (s,z)-Separation (Chapter 4): For Temporal (s,z)-Separation, it was already known that it is NP-hard even on graphs with lifetime two, even if the underlying graph has constant degeneracy and every edge appears in at most one layer [Zsc+20]. We have shown the problem remains NP-hard even if every layer contains only one edge (Proposition 4.1), which we have done by describing how to transform an instance of Temporal (s,z)-Separation to an equivalent instance with this property. However, this should rather be seen as a redistribution of the edges to different time steps and it actually increases the total number of time edges and the lifetime of the instance. For the problem variant Restless Temporal (s,z)-Separation, we have shown that it is Σ_2^P-hard even if every edge appears in at most one layer (Theorem 4.9).
- Temporal Matching (Chapter 5): Here, we have shown that the problem is NP-hard on graphs with lifetime three, even if the underlying graph has maximum degree three and every edge appears in at most one layer (Theorem 5.3). We have further shown that this problem remains NP-hard even if the underlying graph of the input temporal graph is a path (Theorem 5.6).
- (Sliding Window) Temporal Coloring (Chapter 6): For Temporal Coloring, we have shown that it is NP-hard even if the degeneracy of the underlying graph is linear in the number of colors and the number of edges in every layer is quadratic in the number of colors (Theorem 6.4). For the problem variant Sliding Window Temporal Coloring, we have shown that it is NP-hard on graphs with lifetime three, even if the maximum degree of the underlying graph is linear in the number of colors (Theorem 6.5). We have further shown that Sliding Window Temporal Coloring remains NP-hard even if the vertex cover number of the underlying graph is linear in the number of colors (Theorem 6.12). In particular, all mentioned results hold even if the number of colors is two.

We remark that we do not discuss Temporal Clique, Temporal s-Plex, and Temporal Cluster Editing here since their static counterparts are already NP-hard and this thesis does not feature dedicated hardness reductions.

In summary, we can see that for all problems listed above, the reductions we use to show our hardness results create very sparse instances. In particular, this shows that the temporal dimension adds so much complexity to the problem settings that we do not need to create temporal graphs with sophisticated structure in the layers or the underlying graph to encode computationally hard problems. However, as in the static case [ELS13], we know that many real-world instances of temporal graphs are sparse [Ben+19, Him+17], so it is still well-motivated to try to algorithmically exploit sparsity. However, it seems that we have not found many good ways to do that yet.

Adapting existing algorithms is promising. Of course, a (presumably) necessary condition that we need if we want to adapt an existing algorithm to the temporal setting is that the problem which the algorithm solves does not make a computational complexity jump when being transferred to the temporal setting. For most of the problems we discussed in this thesis we have observed a change in the computational complexity, going from being polynomial-time solvable to being NP-hard. However, we successfully adapted existing algorithm for the problems Temporal Clique, Temporal s-Plex, and Temporal Cluster Editing, which are all already NP-hard in the static case. Still, it is necessary to make a number of non-trivial modifications to the algorithms to adapt them for the temporal versions of the problems. In the following, we briefly review the algorithms and how we adapted them.

- Temporal Clique and Temporal s-Plex (Chapter 7): To solve these two problems, we have adapted the famous Bron-Kerbosch algorithm to enumerate cliques in static graphs [BK73]. Our adaptation, as its static role model, is also able to enumerate all maximal temporal cliques and s-plexes as opposed to "just" solving the decision problem. In our adaptation, we mainly had to upgrade the functionality of the data structures used in the algorithm to be able to handle the temporal dimension of the problem. In particular, the notion of neighborhoods had to be adapted to the temporal setting and along with it the data structures to save neighbors or non-neighbors of vertices. Similarly to the static case [ELS13], we were also able to use this algorithm to obtain fixed-parameter tractability for Temporal Clique and Temporal s-Plex (with fixed s) parameterized by the (temporal) degeneracy (Theorem 7.3).

- Temporal Cluster Editing (Chapter 8): For this problem, we have presented a search-tree algorithm showing fixed-parameter tractability for the parameter combination "edit budget" and "marking budget" (Theorem 8.4) and we have shown that the problem admits a polynomial kernel for the parameter combination "edit budget", "marking budget", and "lifetime" (Theorem 8.15).

Our search-tree algorithm is based on a simple branching algorithm to solve Cluster Editing on static graphs that was first described by Gramm et al. [Gra+05] but implicitly already observed by Cai [Cai96]. However, we were not able to deal with the temporal nature of Temporal Cluster Editing without introducing some modifications. In particular, it seemed necessary to start with a greedy localization step, where we (greedily) add and remove edges from the input temporal graph to make all layers "look the same" knowing that we might have to revert some of these greedy decisions later. We also had to introduce a kernelization step within the search-tree algorithm to be able to upper-bound the number of branches in certain situations.

Our kernelization algorithm for Temporal Cluster Editing is based on one of the first kernelization algorithms discovered for Cluster Editing by Gramm et al. [Gra+05]. The main difference was that we cannot simply remove isolated cliques from layers, unless they appear in *every* layer. This alone however was not enough to upper-bound the size of a reduced instance. Hence, we needed a number of additional data reduction rules to handle the temporal nature of the problem.

We remark that our algorithm to enumerate temporal cliques and s-plexes was also evaluated empirically by Bentert et al. [Ben+19] and Himmel et al. [Him+17] showing a good performance in practical applications.

In the literature, we can find more examples where algorithms for static graph problems were successfully adapted to the temporal case, such as algorithms to find temporal paths [Him+19, Wu+16, XFJ03] or algorithms to find *isolated* cliques [MNR19]. We have added two further examples to this list, showing that adaptation of existing algorithms is a canonical and promising way to approach temporal graph problems on an algorithmic level.

Efficient data reduction is difficult. We use the concept of (polynomial) kernelization to analyze the efficiency of data reduction rules in terms of how much they can (provably) reduce the size of an input instance (measured by some parameter). We use the framework of cross-compositions [BJK14, Bod+09, Dru15, Fom+19, FS11] to refute the existence of polynomial kernels based on widely believed complexity assumptions.

For almost all problems considered in this thesis we have shown that they do not admit a polynomial kernel for the number of vertices of the input temporal graph. In other words, for many temporal graph problems it is presumably not possible to design data reduction rules that significantly reduce the lifetime of the input

temporal graph. To give an intuition why this is the case, we give a (very rough) reminder on how cross-compositions work. In a cross-composition, we have to *compose* many problem instances of an NP-hard problem into one big instance of the problem we want to investigate such that the big instance is either a YES-instance if and only if at least one of the input instances is a YES-instance, or if and only if all input instance are YES-instances. This then refutes polynomial kernels for the problem under investigation for parameters of the big instance that only depend on the maximum size of the input instances (and not on the number of input instances).

With temporal graph problems, we have the whole temporal dimension to compose instances. The rough idea for all our results that refute polynomial kernels is as follows. We string the input instances together in the temporal dimension in a clever way, this then results in a large lifetime of the composed instance, but the number of vertices is typically small (that is, only depends on the maximum size of the instances). This rough idea seems to work with many different temporal graph problems.

9.2 Zukunftsmusik

The observations we made in the previous section lead to some general yet canonical future research directions.

- There seems to be much room for new ideas on how to define parameters for temporal graphs, especially ones that measure sparsity. Our results suggest that investigating the structure of the underlying graph is not enough in many cases. Since the complexity of temporal graph problems seems to be "hidden" in the temporal dimension, it would also make sense to explore the idea of measuring the change over time in a temporal graph. To some degree we attempted this successfully with the "shuffle number" (Definition 4.5), which however is a parameter for an already restricted class of temporal graphs. Another example from the literature would be the size of the *temporal core*, that is, the number of vertices adjacent to edges that are not present in all layers. This concept has been successfully employed by Zschoche et al. [Zsc+20] in the context of TEMPORAL (s,z)-SEPARATION.

- The concept of polynomial kernelization does not seem very well-suited for the task of analyzing data reduction rules for temporal graph problems. Our results suggest that concepts such as *partial kernels* [Bet+11] could be more appropriate for this task, where only some dimensions of the input instance are reduced.

Apart from this, we believe that some of our results for TEMPORAL (s,z)-SEPARATION are a first step to an alternative way to approach temporal graph problems by restricting the input instances. In this context we introduced *temporal unit interval graphs* as a very basic model for physical proximity networks and were able to obtain encouraging tractability results on input instances with this restriction. This spawns some canonical future research directions. On the one hand, it would be interesting how far we can generalize the model for temporal physical proximity networks and still obtain tractability results. A first step into this direction would be to consider temporal unit square graphs or temporal unit disk graphs. On the other hand, it makes sense to also investigate other problems under this restriction, especially if they are well-motivated on physical proximity networks. A canonical candidate to start with would be TEMPORAL CLIQUE, since this problem was originally investigated on temporal physical proximity networks [VLM16] and in the static case we know that this problem becomes tractable for example on unit disk intersection graphs [CCJ90].

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009 (cited on pp. 63, 66).

[AF16] K. Axiotis and D. Fotakis. "On the size and the approximability of minimum temporally connected subgraphs". In: *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP '16)*. 2016, 149:1–149:14 (cited on p. 22).

[Akr+17] E. C. Akrida, L. Gąsieniec, G. B. Mertzios, and P. G. Spirakis. "The complexity of optimal design of temporally connected graphs". In: *Theory of Computing Systems* 61(3) (2017), pp. 907–944 (cited on p. 22).

[Akr+19a] E. C. Akrida, J. Czyzowicz, L. Gąsieniec, Ł. Kuszner, and P. G. Spirakis. "Temporal flows in temporal networks". In: *Journal of Computer and System Sciences* 103 (2019), pp. 46–60 (cited on pp. 22, 40).

[Akr+19b] E. C. Akrida, G. B. Mertzios, S. Nikoletseas, C. Raptopoulos, P. G. Spirakis, and V. Zamaraev. "How fast can we reach a target vertex in stochastic temporal graphs?" In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*. Vol. 132. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, 131:1–131:14 (cited on p. 22).

[Akr+20] E. C. Akrida, G. B. Mertzios, P. G. Spirakis, and V. Zamaraev. "Temporal vertex cover with a sliding time window". In: *Journal of Computer and System Sciences* 107 (2020), pp. 108–123 (cited on p. 71).

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993 (cited on p. 39).

[AMS19] E. C. Akrida, G. B. Mertzios, and P. G. Spirakis. "The temporal explorer who returns to the base". In: *Proceedings of the 11th International Conference on Algorithms and Complexity (CIAC '19)*. 2019, pp. 13–24 (cited on p. 22).

[Bam+18] E. Bampis, B. Escoffier, M. Lampis, and V. T. Paschos. "Multistage matchings". In: *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT '18)*. Vol. 101. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 7:1–7:13 (cited on pp. 70, 71, 117).

[Bar16] A.-L. Barabási. *Network Science*. Cambridge University Press, 2016 (cited on p. 21).

[BB13] S. Böcker and J. Baumbach. "Cluster Editing". In: *Proceedings of the 9th Conference on Computability in Europe (CiE '13)*. Vol. 7921. Lecture Notes in Computer Science. Springer, 2013, pp. 33–44 (cited on pp. 145, 146).

[BBC04] N. Bansal, A. Blum, and S. Chawla. “Correlation clustering”. In: *Machine Learning* 56 (2004), pp. 89–113 (cited on pp. 145, 146, 148, 150, 153).

[BBH11] B. Balasundaram, S. Butenko, and I. V. Hicks. “Clique relaxations in social network analysis: The maximum k-plex problem”. In: *Operations Research* 59(1) (2011), pp. 133–142 (cited on pp. 120, 121, 127, 129, 130).

[BBR20] J. Baste, B.-M. Bui-Xuan, and A. Roux. “Temporal matching”. In: *Theoretical Computer Science* 806 (2020), pp. 184–196 (cited on pp. 69, 70, 72, 73, 75, 76, 79, 80).

[BCK15] D. Berlowitz, S. Cohen, and B. Kimelfeld. “Efficient enumeration of maximal k-plexes”. In: *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pp. 431–444 (cited on pp. 120, 121).

[Ben+18] M. Bentert, A.-S. Himmel, H. Molter, M. Morik, R. Niedermeier, and R. Saitenmacher. “Listing all maximal k-plexes in temporal graphs”. In: *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM ’18)*. IEEE Computer Society, 2018, pp. 41–46 (cited on pp. vii, 119).

[Ben+19] M. Bentert, A.-S. Himmel, H. Molter, M. Morik, R. Niedermeier, and R. Saitenmacher. “Listing all maximal k-plexes in temporal graphs”. In: *ACM Journal of Experimental Algorithmics* 24(1) (2019), 13:1–13:27 (cited on pp. vii, 68, 119, 122, 143, 179, 180).

[Ber96] K. A. Berman. “Vulnerability of scheduled networks and a generalization of Menger’s theorem”. In: *Networks: An International Journal* 28(3) (1996), pp. 125–134 (cited on pp. 21, 40).

[Bet+11] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. “Average parameterization and partial kernelization for computing medians”. In: *Journal of Computer and System Sciences* 77(4) (2011), pp. 774–789 (cited on pp. 147, 181).

[Bev+16a] R. van Bevern, T. Fluschnik, G. B. Mertzios, H. Molter, M. Sorge, and O. Suchý. “Finding secluded places of special interest in graphs”. In: *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC ’16)*. Vol. 63. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, 5:1–5:16 (cited on p. viii).

[Bev+16b] R. van Bevern, H. Molter, C. Komusiewicz, R. Niedermeier, M. Sorge, and T. Walsh. “H-Index manipulation by undoing merges”. In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI ’16)*. Vol. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, pp. 895–903 (cited on p. viii).

[Bev+18] R. van Bevern, T. Fluschnik, G. B. Mertzios, H. Molter, M. Sorge, and O. Suchý. “The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs”. In: *Discrete Optimization* 30 (2018), pp. 20–50 (cited on p. viii).

[Bev+20] R. van Bevern, H. Molter, C. Komusiewicz, R. Niedermeier, M. Sorge, and T. Walsh. "H-Index manipulation by undoing merges". In: *Quantitative Science Studies* (2020). Accepted for publication. (cited on p. viii).

[BF14] A. Barrat and J. Fournet. "Contact patterns among high school students". In: *PLoS ONE* 9(9) (2014), e107878:1–e107878:17 (cited on p. 121).

[BFK18] R. van Bevern, V. Froese, and C. Komusiewicz. "Parameterizing edge modification problems above lower bounds". In: *Theory of Computing Systems* 62(3) (2018), pp. 739–770 (cited on pp. 145, 146).

[BG93] J. F. Buss and J. Goldsmith. "Nondeterminism within P". In: *SIAM Journal on Computing* 22(3) (1993), pp. 560–572 (cited on p. 116).

[BJK14] H. L. Bodlaender, B. M. Jansen, and S. Kratsch. "Kernelization lower bounds by cross-composition". In: *SIAM Journal on Discrete Mathematics* 28(1) (2014), pp. 277–305 (cited on pp. 12, 13, 180).

[BK73] C. Bron and J. Kerbosch. "Algorithm 457: Finding all cliques of an undirected graph". In: *Communications of the ACM* 16(9) (1973), pp. 575–577 (cited on pp. 119–122, 130, 131, 179).

[BKZ05] D. Berend, E. Korach, and S. Zucker. "Two-anticoloring of planar and related graphs". In: *Proceedings of the 2005 International Conference on Analysis of Algorithms*. 2005, pp. 335–342 (cited on p. 83).

[BL76] K. S. Booth and G. S. Lueker. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms". In: *Journal of Computer and System Sciences* 13(3) (1976), pp. 335–379 (cited on p. 53).

[Boc+14] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin. "The structure and dynamics of multilayer networks". In: *Physics Reports* 544(1) (2014), pp. 1–122 (cited on pp. 4, 10).

[Bod+09] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. "On problems without polynomial kernels". In: *Journal of Computer and System Sciences* 75(8) (2009), pp. 423–434 (cited on pp. 12, 13, 180).

[BP19] S. Banerjee and B. Pal. "On the enumeration of maximal (Δ, γ)-cliques of a temporal network". In: *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (CoDS-COMAD '19)*. ACM. 2019, pp. 112–120 (cited on p. 121).

[Bre+17] R. Bredereck, C. Komusiewicz, S. Kratsch, H. Molter, R. Niedermeier, and M. Sorge. "Assessing the computational complexity of multi-layer subgraph detection". In: *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC '17)*. Vol. 10236. Lecture Notes in Computer Science. Springer, 2017, pp. 128–139 (cited on p. viii).

[Bre+19] R. Bredereck, C. Komusiewicz, S. Kratsch, H. Molter, R. Niedermeier, and M. Sorge. "Assessing the computational complexity of multilayer subgraph detection". In: *Network Science* 7(2) (2019), pp. 215–241 (cited on pp. viii, 71).

[Buß+20] S. Buß, H. Molter, R. Niedermeier, and M. Rymar. "Algorithmic aspects of temporal betweenness". In: *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*. ACM, 2020, pp. 2084–2092 (cited on p. viii).

[BZ19] H. L. Bodlaender and T. C. van der Zanden. "On exploring always-connected temporal graphs of small pathwidth". In: *Information Processing Letters* 142 (2019), pp. 68–71 (cited on p. 22).

[Cai03] L. Cai. "Parameterized complexity of vertex colouring". In: *Discrete Applied Mathematics* 127(3) (2003), pp. 415–429 (cited on p. 58).

[Cai96] L. Cai. "Fixed-parameter tractability of graph modification problems for hereditary properties". In: *Information Processing Letters* 58(4) (1996), pp. 171–176 (cited on pp. 153, 180).

[Cas+12] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. "Time-varying graphs and dynamic networks". In: *International Journal of Parallel, Emergent and Distributed Systems* 27(5) (2012), pp. 387–408 (cited on pp. 4, 10).

[Cas+15a] A. Casteigts, P. Flocchini, E. Godard, N. Santoro, and M. Yamashita. "On the expressivity of time-varying graphs". In: *Theoretical Computer Science* 590 (2015), pp. 27–37 (cited on p. 21).

[Cas+15b] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. "Shortest, fastest, and foremost broadcast in dynamic networks". In: *International Journal of Foundations of Computer Science* 26(4) (2015), pp. 499–522 (cited on p. 21).

[Cas+20] A. Casteigts, A.-S. Himmel, H. Molter, and P. Zschoche. "The computational complexity of finding temporal paths under waiting time constraints". In: *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC '20)*. LIPIcs. Accepted for publication. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020 (cited on pp. v, vi, 19, 22, 31, 37, 147).

[CC12] Y. Cao and J. Chen. "Cluster Editing: Kernelization based on edge cuts". In: *Algorithmica* 64(1) (2012), pp. 152–169 (cited on pp. 145, 146).

[CCJ90] B. N. Clark, C. J. Colbourn, and D. S. Johnson. "Unit disk graphs". In: *Discrete Mathematics* 86(1-3) (1990), pp. 165–177 (cited on pp. 71, 83, 182).

[CF13a] A. Casteigts and P. Flocchini. *Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics*. Tech. rep. Defence R&D Canada, 2013 (cited on pp. 4, 10).

[CF13b] A. Casteigts and P. Flocchini. *Deterministic Algorithms in Dynamic Networks: Problems, Analysis, and Algorithmic Tools*. Tech. rep. Defence R&D Canada, Apr. 2013 (cited on pp. 4, 10).

[Cha13] G. J. Chang. “Algorithmic aspects of domination in graphs”. In: *Handbook of Combinatorial Optimization* (2013), pp. 221–282 (cited on pp. 82, 83).

[Che+18] J. Chen, H. Molter, M. Sorge, and O. Suchý. “Cluster editing in multi-layer and temporal graphs”. In: *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC ’18)*. Vol. 123. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 24:1–24:13 (cited on pp. viii, 145–148, 150).

[Con+17] A. Conte, D. Firmani, C. Mordente, M. Patrignani, and R. Torlone. “Fast enumeration of large k-plexes”. In: *Proceedings of the 23th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD ’17)*. ACM. 2017, pp. 115–124 (cited on pp. 120, 121).

[Con+18] A. Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari. “D2k: scalable community detection in massive networks via small-diameter k-plexes”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD ’18)*. ACM. 2018, pp. 1272–1281 (cited on pp. 120, 121, 131).

[CPS19] A. Casteigts, J. Peters, and J. Schoeters. “Temporal cliques admit sparse spanners”. In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP ’19)*. Vol. 132. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, 134:1–134:14 (cited on p. 22).

[Cyg+15] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015 (cited on p. 12).

[Deh+04] F. Dehne, M. Fellows, F. Rosamond, and P. Shaw. “Greedy Localization, Iterative Compression, and Modeled Crown Reductions: New FPT Techniques, an Improved Algorithm for Set Splitting, and a Novel $2k$ Kernelization for Vertex Cover”. In: *Proceedings of 1st International Workshop on Parameterized and Exact Computation (IWPEC ’04)*. Vol. 3162. Lecture Notes in Computer Science. Springer, 2004, pp. 271–280 (cited on p. 153).

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013 (cited on p. 12).

[DF95] R. G. Downey and M. R. Fellows. “Parameterized computational feasibility”. In: *Feasible Mathematics II*. Springer, 1995, pp. 219–244 (cited on p. 116).

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999 (cited on pp. 12, 67).

[Die16] R. Diestel. *Graph Theory, 5th Edition*. Vol. 173. Graduate Texts in Mathematics. Springer, 2016 (cited on pp. 9, 74).

[Dör+14] M. Dörnfelder, J. Guo, C. Komusiewicz, and M. Weller. “On the parameterized complexity of consensus clustering”. In: *Theoretical Computer Science* 542 (2014), pp. 71–82 (cited on p. 147).

[DRS17] T. Dey, A. Rossi, and A. Sidiropoulos. “Temporal clustering”. In: *Proceedings of the 25th Annual European Symposium on Algorithms (ESA ’17)*. Vol. 87. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 34:1–34:14 (cited on p. 147).

[Dru15] A. Drucker. “New limits to classical and quantum instance compression”. In: *SIAM Journal on Computing* 44(5) (2015), pp. 1443–1479 (cited on pp. 12, 13, 180).

[EHK15] T. Erlebach, M. Hoffmann, and F. Kammer. “On temporal graph exploration”. In: *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP ’15)*. Vol. 9134. Lecture Notes in Computer Science. Springer, 2015, pp. 444–455 (cited on p. 22).

[ELS13] D. Eppstein, M. Löffler, and D. Strash. “Listing all maximal cliques in large sparse real-world graphs in near-optimal time”. In: *ACM Journal of Experimental Algorithmics* 18(3) (2013), 3.1:1–3.1:21 (cited on pp. 120, 128, 130, 179).

[EMS21] J. Enright, K. Meeks, and F. Skerman. “Changing times to optimise reachability in temporal graphs”. In: *Journal of Computer and System Sciences* 115 (2021), pp. 169–186 (cited on p. 22).

[Enr+19] J. Enright, K. Meeks, G. Mertzios, and V. Zamaraev. “Deleting edges to restrict the size of an epidemic in temporal networks”. In: *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS ’19)*. Vol. 138. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, 57:1–57:15 (cited on p. 22).

[Erl+19] T. Erlebach, F. Kammer, K. Luo, A. Sajenko, and J. T. Spooner. “Two moves per time step make a difference”. In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP ’19)*. Vol. 132. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, 141:1–141:14 (cited on p. 22).

[ES18] T. Erlebach and J. T. Spooner. “Faster exploration of degree-bounded temporal graphs”. In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS ’18)*. Vol. 117. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 36:1–36:13 (cited on p. 22).

[Fel+09] M. R. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. “On the parameterized complexity of multiple-interval graph problems”. In: *Theoretical Computer Science* 410(1) (2009), pp. 53–61 (cited on p. 31).

[Fer+18] H. Fernau, T. Fluschnik, D. Hermelin, A. Krebs, H. Molter, and R. Niedermeier. “Diminishable parameterized problems and strict polynomial kernelization”. In: *Proceedings of the 14th Conference on Computability in Europe (CiE ’18)*. Vol. 10936. Lecture Notes in Computer Science. Springer. 2018, pp. 161–171 (cited on p. viii).

[Fer+20] H. Fernau, T. Fluschnik, D. Hermelin, A. Krebs, H. Molter, and R. Niedermeier. “Diminishable parameterized problems and strict polynomial kernelization”. In: *Computability* 9(1) (2020), pp. 1–24 (cited on p. viii).

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Vol. XIV. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006 (cited on p. 12).

[Flu+17] T. Fluschnik, M. Hatzel, S. Härtlein, H. Molter, and H. Seidler. "The minimum shared edges problem on grid-like graphs". In: *Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG '17)*. Vol. 10520. Lecture Notes in Computer Science. Springer, 2017, pp. 249–262 (cited on p. viii).

[Flu+18] T. Fluschnik, H. Molter, R. Niedermeier, and P. Zschoche. "Temporal graph classes: A view through temporal separators". In: *Proceedings of the 44th International Workshop of Graph-Theoretic Concepts in Computer Science (WG '18)*. Vol. 11159. Lecture Notes in Computer Science. Springer, 2018, pp. 216–227 (cited on pp. vi, 39).

[Flu+19] T. Fluschnik, R. Niedermeier, V. Rohm, and P. Zschoche. "Multistage vertex cover". In: *Proceedings of the 14th International Symposium on Parameterized and Exact Computation (IPEC '19)*. Vol. 148. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, 14:1–14:14 (cited on pp. 71, 117).

[Flu+20a] T. Fluschnik, H. Molter, R. Niedermeier, M. Renken, and P. Zschoche. "As time goes by: reflections on treewidth for temporal graphs". In: *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*. Vol. 12160. Lecture Notes in Computer Science. Springer, 2020, pp. 49–77 (cited on p. viii).

[Flu+20b] T. Fluschnik, H. Molter, R. Niedermeier, M. Renken, and P. Zschoche. "Temporal graph classes: a view through temporal separators". In: *Theoretical Computer Science* 806 (2020), pp. 197–218 (cited on pp. vi, 39, 42).

[FMS13] P. Flocchini, B. Mans, and N. Santoro. "On the exploration of time-varying networks". In: *Theoretical Computer Science* 469 (2013), pp. 53–68 (cited on p. 22).

[Fom+14] F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. "Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters". In: *Journal of Computer and System Sciences* 80(7) (2014), pp. 1430–1447 (cited on pp. 145, 146).

[Fom+19] F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019 (cited on pp. 12, 180).

[FS11] L. Fortnow and R. Santhanam. "Infeasibility of instance compression and succinct PCPs for NP". In: *Journal of Computer and System Sciences* 77(1) (2011), pp. 91–106 (cited on pp. 12, 13, 180).

[GCV91] F. Göbel, J. O. Cerdeira, and H. J. Veldman. "Label-connected graphs and the gossip problem". In: *Discrete Mathematics* 87(1) (1991), pp. 29–40 (cited on p. 21).

[GG15] S. Ghosal and S. C. Ghosh. “Channel assignment in mobile networks based on geometric prediction and random coloring”. In: *Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN ’15)*. IEEE. 2015, pp. 237–240 (cited on p. 90).

[GHN04] J. Guo, F. Hüffner, and R. Niedermeier. “A structural view on parameterizing problems: Distance from triviality”. In: *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC ’04)*. Springer. 2004, pp. 162–173 (cited on p. 58).

[GJ77] M. R. Garey and D. S. Johnson. “The rectilinear Steiner tree problem is NP-complete”. In: *SIAM Journal on Applied Mathematics* 32(4) (1977), pp. 826–834 (cited on p. 81).

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979 (cited on pp. 47, 76, 80, 81, 93, 96, 112).

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. “Some simplified NP-complete problems”. In: *Theoretical Computer Science* 1(3) (1976), pp. 237–267 (cited on pp. 76, 80, 81, 93, 96).

[Gra+05] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. “Graph-modeled data clustering: exact algorithms for clique generation”. In: *Theory of Computing Systems* 38(4) (2005), pp. 373–392 (cited on pp. 145, 146, 153, 159, 167, 168, 175, 180).

[GTW14] A. Gupta, K. Talwar, and U. Wieder. “Changing bases: multistage optimization for matroids and matchings”. In: *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP ’14)*. Vol. 8572. Lecture Notes in Computer Science. Springer. 2014, pp. 563–575 (cited on pp. 70, 71, 117).

[Guo+10] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. “A more relaxed model for graph-based data clustering: s-plex cluster editing”. In: *SIAM Journal on Discrete Mathematics* 24(4) (2010), pp. 1662–1683 (cited on p. 120).

[GZW18] D. Guo, H. Zhang, and M. D. Wong. “On coloring rectangular and diagonal grid graphs for multiple patterning lithography”. In: *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press. 2018, pp. 387–392 (cited on pp. 82, 83).

[Haa+20] R. Haag, H. Molter, R. Niedermeier, and M. Renken. “Feedback edge sets in temporal graphs”. In: *Proceedings of the 46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG ’20)*. Vol. 12301. Lecture Notes in Computer Science. Springer, 2020, pp. 200–212 (cited on p. viii).

[Him+16] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge. “Enumerating maximal cliques in temporal graphs”. In: *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM ’16)*. IEEE Computer Society, 2016, pp. 337–344 (cited on pp. vii, 119).

[Him+17] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge. “Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs”. In: *Social Network Analysis and Mining* 7(1) (2017), 35:1–35:16 (cited on pp. vii, 68, 119, 122, 143, 179, 180).

[Him+19] A.-S. Himmel, M. Bentert, A. Nichterlein, and R. Niedermeier. “Efficient computation of optimal temporal walks under waiting-time constraints”. In: *Proceedings of the 8th International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS '19)*. Vol. 882. SCI. Springer, 2019, pp. 494–506 (cited on pp. 21, 27, 180).

[Him16] A.-S. Himmel. “Enumerating Maximal Cliques in Temporal Graphs”. Bachelor's thesis. TU Berlin, Jan. 2016 (cited on p. vii).

[Him18] A.-S. Himmel. “Algorithmic Investigations into Temporal Paths”. Master's thesis. TU Berlin, Apr. 2018 (cited on pp. v, 21, 27).

[HMS18] C. Hoffmann, H. Molter, and M. Sorge. “The parameterized complexity of centrality improvement in networks”. In: *Proceedings of the 44th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM '18)*. Vol. 10706. Lecture Notes in Computer Science. Springer. 2018, pp. 111–124 (cited on p. viii).

[Hol15] P. Holme. “Modern temporal network theory: a colloquium”. In: *The European Physical Journal B* 88(9) (2015), 234:1–234:30 (cited on pp. 4, 10).

[Hol16] P. Holme. “Temporal network structures controlling disease spreading”. In: *Physical Review E* 94.2 (2016), 022305:1–022305:8 (cited on p. 21).

[HS12] P. Holme and J. Saramäki. “Temporal networks”. In: *Physics Reports* 519(3) (2012), pp. 97–125 (cited on pp. 4, 10, 21).

[HS13] P. Holme and J. Saramäki. *Temporal Networks*. Springer, 2013 (cited on pp. 4, 10).

[HS19] P. Holme and J. Saramäki. *Temporal Network Theory*. Springer, 2019 (cited on pp. 4, 10).

[Hüf+09] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. “Isolation concepts for clique enumeration: comparison and computational experiments”. In: *Theoretical Computer Science* 410(52) (2009), pp. 5384–5397 (cited on p. 120).

[Hun+98] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. “NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs”. In: *Journal of Algorithms* 26(2) (1998), pp. 238–274 (cited on p. 88).

[II09] H. Ito and K. Iwama. “Enumeration of isolated cliques and pseudo-cliques”. In: *ACM Transactions on Algorithms* 5(4) (2009), 40:1–40:21 (cited on p. 120).

[IP01] R. Impagliazzo and R. Paturi. “On the complexity of k-SAT”. In: *Journal of Computer and System Sciences* 62(2) (2001), pp. 367–375 (cited on pp. 14, 30, 31, 107).

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. "Which problems have strongly exponential complexity?" In: *Journal of Computer and System Sciences* 63(4) (2001), pp. 512–530 (cited on pp. 14, 30, 31, 107).

[JT11] T. R. Jensen and B. Toft. *Graph Coloring Problems*. Vol. 39. John Wiley & Sons, 2011 (cited on p. 91).

[Kar72] R. M. Karp. "Reducibility among combinatorial problems". In: *Complexity of Computer Computations*. Springer, 1972, pp. 85–103 (cited on pp. 47, 93, 119, 120, 127, 129, 130).

[Ken38] M. G. Kendall. "A new measure of rank correlation". In: *Biometrika* 30(1/2) (1938), pp. 81–93 (cited on p. 58).

[Kiv+14] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. "Multilayer networks". In: *Journal of Complex Networks* 2(3) (2014), pp. 203–271 (cited on pp. 16, 145).

[KKK02] D. Kempe, J. Kleinberg, and A. Kumar. "Connectivity and inference problems for temporal networks". In: *Journal of Computer and System Sciences* 64(4) (2002), pp. 820–842 (cited on pp. 21, 39–41, 44, 45).

[KL15] J. Kim and J.-G. Lee. "Community detection in multi-layer graphs: A survey". In: *ACM SIGMOD Record* 44(3) (2015), pp. 37–48 (cited on p. 145).

[KM86] M. Křivánek and J. Morávek. "NP-hard problems in hierarchical-tree clustering". In: *Acta informatica* 23(3) (1986), pp. 311–323 (cited on p. 150).

[Kom+09] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. "Isolation concepts for efficiently enumerating dense subgraphs". In: *Theoretical Computer Science* 410(38) (2009), pp. 3640–3654 (cited on p. 120).

[KU12] C. Komusiewicz and J. Uhlmann. "Cluster editing with locally bounded modifications". In: *Discrete Applied Mathematics* 160 (2012), pp. 2259–2270 (cited on pp. 145, 146).

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2(1-2) (1955), pp. 83–97 (cited on p. 151).

[LM17] Q. Liang and E. Modiano. "Survivability in time-varying networks". In: *IEEE Transactions on Mobile Computing* 16(9) (2017), pp. 2668–2681 (cited on p. 40).

[LMS18] J. Luo, H. Molter, and O. Suchý. "A parameterized complexity view on collapsing k-cores". In: *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC '18)*. Vol. 115. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 7:1–7:14 (cited on p. viii).

[LO93] P. J. Looges and S. Olariu. "Optimal greedy algorithms for indifference graphs". In: *Computers & Mathematics with Applications* 25(7) (1993), pp. 15–25 (cited on pp. 43, 53).

[LP09] L. Lovász and M. D. Plummer. *Matching Theory*. Vol. 367. AMS, 2009 (cited on pp. 69, 71, 72).

[Luo+18] J. Luo, H. Molter, A. Nichterlein, and R. Niedermeier. "Parameterized dynamic cluster editing". In: *Proceedings of the 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS '18*. Vol. 122. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 46:1–46:15 (cited on pp. viii, 145, 147).

[Luo+20] J. Luo, H. Molter, A. Nichterlein, and R. Niedermeier. "Parameterized dynamic cluster editing". In: *Algorithmica* (2020), pp. 1–44 (cited on p. viii).

[LVM18] M. Latapy, T. Viard, and C. Magnien. "Stream graphs and link streams for the modeling of interactions over time". In: *Social Network Analysis and Mining* 8(1) (2018), 61:1–61:29 (cited on pp. 4, 10, 74).

[LW70] D. R. Lick and A. T. White. "k-Degenerate graphs". In: *Canadian Journal of Mathematics* 22(5) (1970), pp. 1082–1096 (cited on pp. 121, 128).

[Mat98] T. Matsui. "Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs". In: *Proceedings of the Japanese Conference on Discrete and Computational Geometry (JCDCG '98)*. Springer. 1998, pp. 194–200 (cited on p. 88).

[Men27] K. Menger. "Zur allgemeinen Kurventheorie". German. In: *Fundamenta Mathematicae* 10(1) (1927), pp. 96–115 (cited on p. 40).

[Mer+20] G. B. Mertzios, H. Molter, R. Niedermeier, V. Zamaraev, and P. Zschoche. "Computing maximum matchings in temporal graphs". In: *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS '20)*. Vol. 154. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 27:1–27:14 (cited on pp. vi, 69, 72, 75, 79–81, 88).

[MH12] B. McClosky and I. V. Hicks. "Combinatorial algorithms for the maximum k-plex problem". In: *Journal of Combinatorial Optimization* 23(1) (2012), pp. 29–49 (cited on pp. 120, 121).

[Mic16] O. Michail. "An introduction to temporal graphs: an algorithmic perspective". In: *Internet Mathematics* 12(4) (2016), pp. 239–280 (cited on pp. 4, 10).

[Mil+18] M. G. Millani, H. Molter, R. Niedermeier, and M. Sorge. "Efficient algorithms for measuring the funnel-likeness of DAGs". In: *Proceedings of the 5th International Symposium on Combinatorial Optimization (ISCO '18)*. Vol. 10856. Lecture Notes in Computer Science. Springer. 2018, pp. 183–195 (cited on p. viii).

[Mil+20] M. G. Millani, H. Molter, R. Niedermeier, and M. Sorge. "Efficient algorithms for measuring the funnel-likeness of DAGs". In: *Journal of Combinatorial Optimization* 39(1) (2020), pp. 216–245 (cited on p. viii).

[MMS19] G. B. Mertzios, O. Michail, and P. G. Spirakis. “Temporal network optimization subject to connectivity constraints”. In: *Algorithmica* 81(4) (2019), pp. 1416–1449 (cited on pp. 22, 41).

[MMZ19] G. B. Mertzios, H. Molter, and V. Zamaraev. “Sliding window temporal graph coloring”. In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI ’19)*. AAAI Press, 2019, pp. 7667–7674 (cited on pp. vii, 89, 92, 111).

[MNR19] H. Molter, R. Niedermeier, and M. Renken. “Enumerating isolated cliques in temporal networks”. In: *Proceedings of the 8th International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS ’19)*. Vol. 882. Studies in Computational Intelligence. Springer, 2019, pp. 519–531 (cited on pp. viii, 121, 180).

[MNR20] H. Molter, R. Niedermeier, and M. Renken. “Isolation concepts applied to temporal clique enumeration”. In: *Network Science* (2020), pp. 1–23 (cited on p. viii).

[MNS12] H. Moser, R. Niedermeier, and M. Sorge. “Exact combinatorial algorithms and experiments for finding maximum k-plexes”. In: *Journal of Combinatorial Optimization* 24(3) (2012), pp. 347–373 (cited on pp. 120, 121).

[MS16] O. Michail and P. G. Spirakis. “Traveling salesman problems in temporal graphs”. In: *Theoretical Computer Science* 634 (2016), pp. 1–23 (cited on pp. 22, 71).

[MV80] S. Micali and V. V. Vazirani. “An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs”. In: *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS ’80)*. IEEE. 1980, pp. 17–27 (cited on pp. 71, 74).

[MZZ16] C. Ma, T. Zhou, and H.-F. Zhang. “Playing the role of weak clique property in link prediction: A friend recommendation model”. In: *Scientific Reports* 6 (2016), 30098:1–30098:11 (cited on p. 120).

[Nas61] C. S. J. A. Nash-Williams. “Edge-disjoint spanning trees of finite graphs”. In: *Journal of the London Mathematical Society* 1(1) (1961), pp. 445–450 (cited on p. 129).

[Nas64] C. S. J. A. Nash-Williams. “Decomposition of finite graphs into forests”. In: *Journal of the London Mathematical Society* 1(1) (1964), pp. 12–12 (cited on p. 129).

[ND12] J. Nešetřil and P. O. De Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer, 2012 (cited on p. 37).

[New18] M. E. J. Newman. *Networks*. Oxford University Press, 2018 (cited on p. 21).

[Nic+13] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. “Graph metrics for temporal networks”. In: *Temporal networks*. Springer, 2013, pp. 15–40 (cited on p. 120).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cited on p. 12).

[PM81] A. Paz and S. Moran. “Non deterministic polynomial optimization problems and their approximations”. In: *Theoretical Computer Science* 15(3) (1981), pp. 251–277 (cited on p. 67).

[Pol74] S. Poljak. “A note on stable sets and colorings of graphs”. In: *Commentationes Mathematicae Universitatis Carolinae* 15(2) (1974), pp. 307–309 (cited on p. 87).

[PS11] R. K. Pan and J. Saramäki. “Path lengths, correlations, and centrality in temporal networks”. In: *Physical Review E* 84(1) (2011), 016105:1–016105:10 (cited on p. 21).

[PYB13] J. Pattillo, N. Youssef, and S. Butenko. “On clique relaxation models in network analysis”. In: *European Journal of Operational Research* 226(1) (2013), pp. 9–18 (cited on pp. 120, 121).

[Qin+19] H. Qin, R.-H. Li, G. Wang, L. Qin, Y. Cheng, and Y. Yuan. “Mining periodic cliques in temporal networks”. In: *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE ’19)*. IEEE. 2019, pp. 1130–1141 (cited on p. 121).

[Sch78] T. J. Schaefer. “The complexity of satisfiability problems”. In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC ’78)*. ACM. 1978, pp. 216–226 (cited on p. 112).

[Sch98] A. Schrijver. “Bipartite edge coloring in $O(\Delta m)$ time”. In: *SIAM Journal on Computing* 28(3) (1998), pp. 841–846 (cited on p. 76).

[SF78] S. B. Seidman and B. L. Foster. “A graph-theoretic generalization of the clique concept”. In: *Journal of Mathematical Sociology* 6(1) (1978), pp. 139–154 (cited on pp. 120, 121, 127).

[SST04] R. Shamir, R. Sharan, and D. Tsur. “Cluster graph modification problems”. In: *Discrete Applied Mathematics* 144(1-2) (2004), pp. 173–182 (cited on pp. 145, 146, 148, 150, 175).

[Sto76] L. J. Stockmeyer. “The polynomial-time hierarchy”. In: *Theoretical Computer Science* 3(1) (1976), pp. 1–22 (cited on pp. 63, 66).

[SW19] M. Sorge and M. Weller. *The graph parameter hierarchy (manuscript)*. 2019. URL: https://manyu.pro/assets/parameter-hierarchy.pdf (cited on pp. 15, 129).

[TAG17] A. Tagarelli, A. Amelio, and F. Gullo. “Ensemble-based community detection in multilayer networks”. In: *Data Mining and Knowledge Discovery* 31(5) (2017), pp. 1506–1543 (cited on p. 145).

[TB11] C. Tantipathananandh and T. Y. Berger-Wolf. “Finding Communities in Dynamic Social Networks”. In: *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM ’11)*. IEEE Computer Society, 2011, pp. 1236–1241 (cited on pp. 145–147, 150).

[TBK07] C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe. “A Framework for Community Identification in Dynamic Social Networks”. In: *Proceedings of the 13th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07). ACM, 2007, pp. 717–726 (cited on pp. 145–147, 150).

[TLD09] W. Tang, Z. Lu, and I. S. Dhillon. "Clustering with Multiple Graphs". In: *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM '09)*. IEEE Computer Society, 2009, pp. 1016–1021 (cited on p. 145).

[Tov84] C. A. Tovey. "A simplified NP-complete satisfiability problem". In: *Discrete Applied Mathematics* 8(1) (1984), pp. 85–89 (cited on pp. 27, 28, 30, 97, 102, 107, 111, 112).

[TTT06] E. Tomita, A. Tanaka, and H. Takahashi. "The worst-case time complexity for generating all maximal cliques and computational experiments". In: *Theoretical Computer Science* 363(1) (2006), pp. 28–42 (cited on p. 120).

[Tut61] W. T. Tutte. "On the problem of decomposing a graph into n connected factors". In: *Journal of the London Mathematical Society* 1(1) (1961), pp. 221–230 (cited on p. 129).

[Val81] L. G. Valiant. "Universality considerations in VLSI circuits". In: *IEEE Transactions on Computers* 100(2) (1981), pp. 135–140 (cited on pp. 83, 84).

[Viz15] V. G. Vizing. "On coloring problems for two-season multigraphs". In: *Journal of Applied and Industrial Mathematics* 9(2) (2015), pp. 292–296 (cited on p. 90).

[Viz64] V. G. Vizing. "On an estimate of the chromatic class of a p-graph". In: *Diskretnyi Analiz* 3 (1964), pp. 25–30 (cited on p. 76).

[VLM16] T. Viard, M. Latapy, and C. Magnien. "Computing maximal cliques in link streams". In: *Theoretical Computer Science* 609 (2016), pp. 245–252 (cited on pp. 68, 119–121, 126, 127, 143, 182).

[VML18] T. Viard, C. Magnien, and M. Latapy. "Enumerating maximal cliques in link streams with durations". In: *Information Processing Letters* 133 (2018), pp. 44–48 (cited on pp. 121, 143).

[WP07] B. Wu and X. Pei. "A parallel algorithm for enumerating all the maximal k-plexes". In: *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '07)*. Vol. 4819. Lecture Notes in Computer Science. Springer. 2007, pp. 476–483 (cited on pp. 120, 121, 131).

[Wu+16] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. "Efficient algorithms for temporal path computation". In: *IEEE Transactions on Knowledge and Data Engineering* 28(11) (2016), pp. 2927–2942 (cited on pp. 19, 21, 180).

[XFJ03] B. B. Xuan, A. Ferreira, and A. Jarry. "Computing shortest, fastest, and foremost journeys in dynamic networks". In: *International Journal of Foundations of Computer Science* 14(02) (2003), pp. 267–285 (cited on pp. 19, 21, 43, 180).

[Xia+17] M. Xiao, W. Lin, Y. Dai, and Y. Zeng. "A fast algorithm to compute maximum k-plexes in social network analysis". In: *Proceedings of the 31st AAAI Conference on*

Artificial Intelligence (AAAI '17). AAAI Press, 2017, pp. 919–925 (cited on pp. 120, 121).

[Yu+13] F. Yu, A. Bar-Noy, P. Basu, and R. Ramanathan. "Algorithms for channel assignment in mobile wireless networks using temporal coloring". In: *Proceedings of the 16th ACM International Conference on Modeling, Analysis & Simulation of Wireless and Mobile Systems (MSWiM '13)*. ACM. 2013, pp. 49–58 (cited on p. 90).

[Zsc+18] P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. "The complexity of finding separators in temporal graphs". In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS '18)*. Vol. 117. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 45:1–45:17 (cited on pp. vi, 39).

[Zsc+20] P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. "The complexity of finding separators in temporal graphs". In: *Journal of Computer and System Sciences* 107 (2020), pp. 72–92 (cited on pp. vi, 27, 39, 41, 44, 45, 47, 49, 67, 178, 181).

[Zsc17] P. Zschoche. "On Finding Separators in Temporal Graphs". Master's thesis. TU Berlin, Aug. 2017 (cited on pp. v, vi, 21, 67).

Problem Compendium

Temporal Graph Problems

For easy reference, we list (in alphabetical order) below the problem definitions of the temporal graph problems that are under consideration in this thesis.

RESTLESS TEMPORAL (s,z)-PATH

Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, two distinct vertices $s, z \in V$, and an integer $\Delta \leq \ell$.

Question: Is there a Δ-restless temporal path from s to z in $\mathcal{G}$?

Chapter 3

RESTLESS TEMPORAL (s,z)-SEPARATION

Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, two distinct vertices $s, z \in V$, and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.

Question: Does $\mathcal{G}$ admit a Δ-restless temporal (s,z)-separator of size at most k?

Chapter 4

SHORT RESTLESS TEMPORAL (s,z)-PATH

Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, two distinct vertices $s, z \in V$, and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.

Question: Is there a Δ-restless temporal path of length at most k from s to z in $\mathcal{G}$?

Chapter 3

SLIDING WINDOW TEMPORAL COLORING

Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.

Question: Is there a proper sliding Δ-window temporal coloring Υ of $\mathcal{G}$ using $|\Upsilon| \leq k$ colors?

Chapter 6

TEMPORAL CLIQUE
Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and three integers $k, t \in \mathbb{N}$, and $\Delta \leq \ell$.
Question: Does $\mathcal{G}$ contain a Δ-clique $(C, [a, b])$ with $|C| \geq k$ and $b - a \geq t$?

Chapter 7

TEMPORAL CLUSTER EDITING
Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers $k, d \in \mathbb{N}$.
Question: Is there a d-consistent k-bounded clustering for $\mathcal{G}$?

Chapter 8

TEMPORAL COLORING
Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and an integer $k \in \mathbb{N}$.
Question: Is there a proper temporal coloring Υ of $\mathcal{G}$ using $|\Upsilon| \leq k$ colors?

Chapter 6

TEMPORAL MATCHING
Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and two integers $k \in \mathbb{N}$ and $\Delta \leq \ell$.
Question: Is there a Δ-temporal matching in $\mathcal{G}$ of cardinality at least k?

Chapter 5

TEMPORAL s-PLEX
Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$ and three integers $k, t \in \mathbb{N}$, and $\Delta \leq \ell$.
Question: Does $\mathcal{G}$ contain a Δ-s-plex $(C, [a, b])$ with $|C| \geq k$ and $b - a \geq t$?

Chapter 7

TEMPORAL (s, z)-SEPARATION
Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[\ell]})$, two distinct vertices $s, z \in V$, and an integer $k \in \mathbb{N}$.
Question: Does $\mathcal{G}$ admit a temporal (s, z)-separator of size at most k?

Chapter 4

Non-Temporal Problems

We list (in alphabetical order) below the problem definitions of problems that are mentioned or used for reductions in this thesis.

∃∀-SAT

Input: A Boolean formula ϕ in conjunctive normal form and a partitioning of the variables of ϕ into two sets X and Y.

Question: Is there an assignment of truth values for the variables in X such that, for all possible assignments of truth values for the variables in Y, the formula ϕ evaluates to true?

3-SAT

Input: A Boolean formula ϕ in conjunctive normal form where every clause has three literals.

Question: Is ϕ satisfiable?

4-COLORING

Input: An undirected graph $G = (V, E)$.

Question: Is there a vertex coloring function $\Upsilon : V \to [4]$ such that for all $\{v, w\} \in E$ it holds that $\Upsilon(v) \neq \Upsilon(w)$?

CLIQUE

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Question: Is there a vertex set $V' \subseteq V$ with $|V'| \geq k$ such that for all $v, w \in V'$ it holds that $\{v, w\} \in E$?

CLUSTER EDITING

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Question: Is there an edge modification set $M \subseteq \binom{V}{2}$ with $|M| \leq k$ such that $G' = (V, E \oplus M)$ is a cluster graph?

COLORING

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Question: Is there a vertex coloring function $\Upsilon : V \to [k]$ such that for all $\{v, w\} \in E$ it holds that $\Upsilon(v) \neq \Upsilon(w)$?

EXACT $(3,4)$-SAT

Input: A Boolean formula ϕ in conjunctive normal form where every clause has exactly three distinct literals and every variable appears in exactly four clauses.
Question: Is ϕ satisfiable?

HITTING SET

Input: A universe set U, a family of sets $S_1, \ldots, S_m \subseteq U$, and an integer $k \in \mathbb{N}$.
Question: Is is a set $S^\star \subseteq U$ with $|S^\star| \leq k$ such that for all $i \in [m]$ we have that $S^\star \cap S_i \neq \emptyset$?

INDEPENDENT SET

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Is there a vertex set $V' \subseteq V$ with $|V'| \geq k$ such that for all $v, w \in V'$ it holds that $\{v, w\} \notin E$?

MAXIMUM MATCHING

Input: An undirected graph $G = (V, E)$.
Output: A set $M \subseteq E$ of maximum cardinality such that for all $e, e' \in M$ with $e \neq e'$ it holds that $e \cap e' = \emptyset$.

MONOTONE EXACTLY 1-IN-3 SAT

Input: A collection of triples of variables.
Question: Is there an assignment of truth values for the variables such that exactly one variable of each triple is set to true?

MULTICOLORED CLIQUE

Input: An undirected k-partite graph $G = (V_1 \uplus V_2 \uplus \ldots \uplus V_k, E)$.
Question: Is there a vertex set $V' \subseteq \bigcup_{i \in [k]} V_i$ with $|V'| = k$ such that for all $v, w \in V'$ it holds that $\{v, w\} \in E$?

s-PLEX

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Is there a vertex set $V' \subseteq V$ with $|V'| \geq k$ such that for all $v \in V'$ it holds that $|N(v) \cap V'| \geq |V'| - s$?

VERTEX COVER	
Input:	An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question:	Is there a vertex set $V' \subseteq V$ with $\lvert V' \rvert \leq k$ such that for all $\{v, w\} \in E$ it holds $\{v, w\} \cap V' \neq \emptyset$?

Index

Schriftenreihe **Foundations of computing**
Hrsg.: Prof. Dr. Stephan Kreutzer, Prof. Dr. Uwe Nestmann, Prof. Dr. Rolf Niedermeier

ISSN 2199-5249 (print)
ISSN 2199-5257 (online)

01: Bevern, René van: Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications. - 2014. - 225 S.
ISBN **978-3-7983-2705-4** (print) EUR **12,00**
ISBN **978-3-7983-2706-1** (online)

02: Nichterlein, André: Degree-Constrained Editing of Small-Degree Graphs. - 2015. - xiv, 225 S.
ISBN **978-3-7983-2705-4** (print) EUR **12,00**
ISBN **978-3-7983-2706-1** (online)

03: Bredereck, Robert: Multivariate Complexity Analysis of Team Management Problems. - 2015. - xix, 228 S.
ISBN **978-3-7983-2764-1** (print) EUR **12,00**
ISBN **978-3-7983-2765-8** (online)

04: Talmon, Nimrod: Algorithmic Aspects of Manipulation and Anonymization in Social Choice and Social Networks. - 2016. - xiv, 275 S.
ISBN **978-3-7983-2804-4** (print) EUR **13,00**
ISBN **978-3-7983-2805-1** (online)

05: Siebertz, Sebastian: Nowhere Dense Classes of Graphs. Characterisations and Algorithmic Meta-Theorems. - 2016. - xxii, 149 S.
ISBN **978-3-7983-2818-1** (print) EUR **11,00**
ISBN **978-3-7983-2819-8** (online)

06: Chen, Jiehua: Exploiting Structure in Computationally Hard Voting Problems. - 2016. - xxi, 255 S.
ISBN **978-3-7983-2825-9** (print) EUR **13,00**
ISBN **978-3-7983-2826-6** (online)

07: Arbach, Youssef: On the Foundations of dynamic coalitions. Modeling changes and evolution of workflows in healthcare scenarios. - 2016. - xv, 171 S.
ISBN **978-3-7983-2856-3** (print) EUR **12,00**
ISBN **978-3-7983-2857-0** (online)

08: Sorge, Manuel: Be sparse! Be dense! Be robust! Elements of parameterized algorithmics. - 2017. - xvi, 251 S.
ISBN **978-3-7983-2885-3** (print) EUR **13,00**
ISBN **978-3-7983-2886-0** (online)

09: Dittmann, Christoph: Parity games, separations, and the modal μ-calculus. - 2017. - x, 274 S.
ISBN **978-3-7983-2887-7** (print) EUR **13,00**
ISBN **978-3-7983-2888-4** (online)

10: Karcher, David S.: Event Structures with Higher-Order Dynamics. - 2019. - xix, 125 S.
ISBN **978-3-7983-2995-9** (print) EUR **11,00**
ISBN **978-3-7983-2996-6** (online)

11: Jungnickel, Tim: On the Feasibility of Multi-Leader Replication in the Early Tiers. - 2018. - xiv, 177 S.
ISBN **978-3-7983-3001-6** (print) EUR **13,00**
ISBN **978-3-7983-3002-3** (online)

12: Froese, Vincent: Fine-Grained Complexity Analysis of Some Combinatorial Data Science Problems. - 2018. - xiv, 166 S.
ISBN **978-3-7983-3003-0** (print) EUR **11,00**
ISBN **978-3-7983-3004-7** (online)

Universitätsverlag der TU Berlin

Classic Graph Problems Made Temporal – A Parameterized Complexity Analysis

This thesis investigates the parameterized computational complexity of six classic graph problems lifted to a temporal setting. More specifically, we consider problems defined on temporal graphs, that is, a graph where the edge set may change over a discrete time interval, while the vertex set remains unchanged. Temporal graphs are well-suited to model dynamic data and hence they are naturally motivated in contexts where dynamic changes or time-dependent interactions play an important role, such as, for example, communication networks, social networks, or physical proximity networks. The most important selection criteria for our problems was that they are well-motivated in the context of dynamic data analysis.
Since temporal graphs are mathematically more complex than static graphs, it is maybe not surprising that all problems we consider in this thesis are NP-hard. We focus on the development of exact algorithms, where our goal is to obtain fixed-parameter tractability results, and refined computational hardness reductions that either show NP-hardness for very restricted input instances or parameterized hardness with respect to “large” parameters.

ISBN 978-3-7983-3172-3 (print)
ISBN 978-3-7983-3173-0 (online)

ISBN 978-3-7983-3172-3

http://verlag.tu-berlin.de