The University of Liverpool

Doctoral Thesis

Temporal Graphs

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Networks and Distributed Computing Group
Algorithms Section
Department of Computer Science

by

Eleni C. Akrida

November 14, 2016

"If knowledge can create problems, it is not through ignorance that we can solve them."

Isaac Asimov

THE UNIVERSITY OF LIVERPOOL

Abstract

Faculty of Science and Engineering
Department of Computer Science

Doctor of Philosophy

Temporal Graphs

by Eleni C. Akrida

This thesis studies Temporal Graphs, also called Temporal Networks. More specifically, the project aimed to carry out research on the properties of Temporal Graphs, both in general and in specific classes of the model, as well as examine and develop algorithms for solving problems in temporal graph theory. Temporal graphs are graphs that change -often dramatically- as time progresses, however maintaining a fixed number of vertices. We investigate a range of different temporal graph models depending on the way these changes occur, e.g., in a deterministic or probabilistic fashion, in a discrete-time or continuous-time context, etc. In particular, we examine connectivity matters in a model of temporal graphs where changes happen at random discrete moments in time. Within this framework, we also investigated a model of continuous time and developed algorithms that solve connectivity problems in that model. Furthermore, we study temporal network design issues for the discrete-time model of temporal graphs, both in cases where changes happen deterministically and in cases where changes happen at random discrete-time moments. We also introduce and investigate temporal network flows, where we define the problem of computing a maximum flow in a given temporal network and discuss efficient ways of solving it.

Biographical Sketch

Eleni Akrida was born to Christos Akridas and Eirini Tsoni on November 2, 1988, in Patras, Greece. She attended the 40th Elementary School, the 16th Junior High School and the 1st High School of Patras, where she graduated in 2006.

Eleni enrolled in the Mathematics department of the University of Patras, where she graduated from in 2010 with First-class Honours. She then started her Master's degree in the Mathematics department and the Computer Engineering and Informatics Department of the University of Patras, splitting her time between her studies and her singing. Eleni sang professionally for 3 years while she was also continuing her postgraduate studies, until she obtained her M.Sc. in 2013. After graduating, she moved to Liverpool to pursue a Ph.D. in Computer Science at the University of Liverpool. While in Liverpool, she also started training for her first Marathon following in the footsteps of her dad, who was a marathon runner himself. She has now finished two half-Marathons and a Marathon.

Upon completion of her Ph.D., Eleni plans to pursue a career in academia, and not stop running (literally and figuratively).

Acknowledgements

I am deeply grateful to my supervisor, Paul Spirakis, whose guidance, knowledge and patience added greatly to my experience as a Ph.D. student. Throughout my academic and personal journey during my time as your student, you have always been generous in sharing your time, insights and great experience with me. Thank you for always finding ways to motivate me and inspire me to keep doing research, and for believing in me ever since I met you.

I am especially thankful to my second supervisor, Leszek Gąsieniec, with whom I also worked closely during the course of my Ph.D.. Thank you for acquainting me to collaborators, for guiding me and sharing with me your extensive knowledge, your insight and your creative thinking.

Many thanks also go to my academic advisors, Darek Kowalski and Piotr Krysta, for their time and feedback during my studies. I must also acknowledge all of my co-authors and collaborators for their contributions to the papers we worked on together. I found those experiences both enjoyable and educating.

My three years at the University of Liverpool were funded by the School of Electrical Engineering, Electronics and Computer Science. I very much appreciate this financial support, as it allowed me to focus on my studies and personal interests. Special thanks also goes to the NeST initiative of the School of Electrical Engineering, Electronics and Computer Science.

For the fruitful conversations, research-wise or not, and for the fun experiences, I wish to thank all my friends and colleagues from the Department of Computer Science at the University of Liverpool; thank you to my officemates for all the laughs and countless lunches we had together, without which my years as a Ph.D. student would have certainly been tougher and less colourful. Special thanks also go to Thomas Gorry for all his love and support – my Ph.D. journey would not have been the same without you.

Finally, thanks to my family, my Mum, my Dad, my sister Evina and my brother Giorgos – you have been a great support and encouragement throughout my life. To my Mum and Dad, thank you for being strict when I was little and supporting my decisions when I grew up; thank you for introducing the beauty of maths to me and for always insisting I do my best.

Contents

List of Figures

List of Algorithms

List of Tables

Abbreviations

iff	**IF** and only I**F**
whp	**W**ith **H**igh **P**robability
resp.	**RESP**ectively
s.t.	**S**uch **T**hat
w.r.t.	**W**ith **R**espect **T**o
PTAS	**P**olynomial-**T**ime **A**pproximation **S**cheme

Dedicated to my parents, Christos and Eirini, my sister Evina, and my brother Giorgos.

Thank you for supporting and guiding me to where I am today.

Chapter 1

Introduction and our Results

Graph theory is a branch of Mathematics and Computer Science, concerned with networks of nodes connected by links. Within this thesis, we will be alternating between the terms *graph* and *network* to describe the same notion: a group of points, representing the participating entities of the network, connected by links that represent the communication opportunities between the points. One can trace the origins of graph theory back in 1735, when the Swiss mathematician Leonhard Euler solved the Königsberg bridge problem. The latter was an old puzzle concerning whether there existed a walk through the city of Königsberg in Prussia that crossed each of the city's seven bridges once and only once. The development of graph theory through the years has been very similar to that of probability theory, in the sense that much of the original work in both areas was motivated by the study of games and recreational mathematics. Since then, the study and analysis of (algorithms on) graphs has evolved into a major research topic not only in Computer Science, but across a wide range of Science and Engineering. That is due to the appealing property of graphs being that they can be used to model several kinds of relations and processes in physical, social, biological and information systems. However, in the ever-changing modern society with the highly dynamic nature of interactions, representing such systems as static graphs can often be restricting. For example, wired or wireless networks may change dynamically, transport network connections may only operate at certain times, friendships on social media may be added and removed over time etc. Basic notions and concepts of static graph theory need to be revisited when studying such dynamically changing systems; for example, the idea of (static) connectivity of a network may be ambiguous in the temporal, i.e. dynamically changing, context, since the connectivity of nodes now crucially depends on the appearance of the network's links over time. It may be that in a time-varying network, the static networks that are formed at *every* time step are disconnected, but the network itself is *temporally* connected, meaning that any participating entity may be able to communicate with any other participating entity over time.

A *temporal network* or *temporal graph* is a triplet $G(L) = (V, E, L)$, where $G = (V, E)$ is an underlying (di)graph and L is a time-labelling assignment that assigns to each edge of G a set of discrete time-labels. These labels are natural numbers that indicate the discrete moments in time at which each edge becomes available, i.e., exists and can be used as a connection between its endpoints. This discrete-time modelling allows for better investigation of temporal graphs from an algorithmic point of view and gives a purely combinatorial flavour to the structure and the arising problems. It is also a natural abstraction of several real-life scenarios and systems that operate in discrete time instances, e.g., synchronous mobile distributed systems that operate in discrete rounds.

Throughout the chapters, there are some definitions that remain fixed. We present those definitions here for the reader to refer to them if needed, but we also define the respective notions in each individual chapter, where appropriate.

Definition 1.1 (**Temporal Graph**)**.** Let $G = (V, E)$ be a (di)graph. A temporal graph (or temporal network) on G is an ordered triple $G(L) = (V, E, L)$, where $L = \{L_e \subseteq \mathbb{N} : e \in E\}$ is an *assignment* of labels to the edges of G.

Any implicit definition of a dynamic graph can eventually be modified to extract a temporal graph or temporal graph patterns. Consider, for example, the model of Markovian (q, p) graphs: In this model of dynamic random graphs [32], each edge at every time step can be in one of two states, ON or OFF, and the probability distribution is governed by a two-state Markov chain. The transition probabilities are given by $Pr(OFF \rightarrow ON) = p$, $Pr(OFF \rightarrow OFF) = 1 - p$, $Pr(ON \rightarrow OFF) = q$, and $Pr(ON \rightarrow ON) = 1 - q$. Now, starting from an initial graph at time 0, the above definition constructs "branches" of states of the graph at subsequent time steps, each branch occuring with some probability. Every such branch can be viewed as a temporal graph.

Different authors may define the notion of a temporal graph differently, but the idea remains the same; it is a graph in which the edge set can change in every (time) step.

Each edge, together with a label that is assigned to it, defines a time-edge. Time-edges show the communicating opportunities over time between each pair of vertices, as well as a direction to the way the communication may happen, e.g., which vertex may pass information to which other vertex.

Definition 1.2 (**Time edge**). Let $e = \{u, v\}$ (resp. $e = (u, v)$) be an edge (resp. arc) of the underlying (di)graph of a temporal graph and consider a label $l \in L_e$. The ordered triplet (u, v, l), also denoted as (e, l), is called *time edge*. We denote the set of time edges of a temporal graph $G(L)$ by E_L.

Note that an undirected edge $e = \{u, v\}$ is associated with $2 \cdot |L_e|$ time edges, namely both (u, v, l) and (v, u, l) for every $l \in L_e$.

In the broader picture, we are interested in the way that information may be transferred not only from one vertex to another vertex in a single time step, but also *over time*, i.e., over several time steps. For that reason, we define the notion of a journey, which is, roughly speaking, a path over time in a temporal graph.

Definition 1.3 (**Journey**). A *temporal path* or *journey* j from a vertex u to a vertex v (*$u \to v$ journey*) is a sequence of time edges (u, u_1, l_1), (u_1, u_2, l_2), $\ldots$, (u_{k-1}, v, l_k), such that $l_i < l_{i+1}$, for each $1 \leq i \leq k-1$. We call the last time label of journey j, l_k, *a*rrival time of the journey.

We are interested in how *early* (in time) information may be transferred, so we are interested in what we call *foremost* journeys. Note here that other authors have also defined the notions of *shortest* and *fastest* journeys (see Section 1.1) as journeys with the smallest number of edges, and the smallest duration, respectively, but for the purpose of this Thesis we focus on foremost journeys:

Definition 1.4 (**Foremost journey**). A $u \to v$ journey in a temporal graph is called *foremost journey* if its arrival time is the minimum arrival time of all $u \to v$ journeys' arrival times, under the labels assigned to the underlying graph's edges.

Definition 1.5 (**Temporal distance**). The *temporal distance* of a (target) vertex v from a (source) vertex u is the arrival time of the foremost $u \to v$ journey and is denoted by $\delta(u, v)$.

This thesis goes towards the development of an algorithmic temporal graph theory that is analogous to the static graph theory. The novelty of this direction lies in the fact that when looking into temporal graph problems, one often needs to employ techniques that are entirely different to those used to tackle corresponding static graph problems, as time is introduced as another “dimension” of the graph and an additional factor to the problem. To better see the effect that time has as a parameter of the structure, think of how the various patterns in which the time-labels are assigned to the edges affect the computational complexity of the problems that we may be concerned with; this is a concept that has no straightforward equivalent in static graphs.

1.1 Background

Dynamic graphs in general have received a growing interest in recent years, since they can model several real-life scenarios of our fast-changing modern world; temporal graphs are a type of such dynamic graphs. The study of temporal graphs is quite interdisciplinary, which is reflected in literature where the notion is studied under various names, such as temporal graphs [6, 42], temporal networks [4, 8], evolving graphs [21, 25, 32, 44], graphs over time [79] or time-varying graphs [1, 29, 47, 97]. However, the term “dynamic” has been also used in literature to describe other notions of dynamicity over time. Fleisher and Skutella [45], as well as Hoppe and Tardos [65], study flows over time on *their* notion of dynamic networks, that is static graphs with transit times on their edges. Demetrescu et al. [35] consider dynamic graphs as graphs that are subject to updates, but their notion of a dynamic graph algorithm involves updating the solution of a problem efficiently after these updates, instead of recomputing it from scratch. In temporal graphs, however, the graph topology is expected to change so immensely that the changes are no longer considered an anomaly, but rather an inherent property of the network itself.

Looking at temporal graphs, one can quickly realise that the introduction of dynamicity as a built-in property of the graph has an impact on basic graph parameters and metrics. For example, for the graph to be connected, global connectivity at every step in time may no longer be necessary; one may consider connectivity *over time*, i.e., existence of communication routes over time between all pairs of nodes. A *journey* (or *temporal path* or *time-respecting path*) is exactly that: a path in the underlying graph, with edges that appear in an increasing order over time [69]. The known metrics of static graphs usually have temporal graph analogues, depending on the particular question posed; for example, the notion of a shortest path in a static graph may translate into a temporal path with the *smallest number of edges* (shortest journey), or the *smallest duration* (fastest journey), or the *smallest arrival time* (foremost journey) [25, 69, 81]. Wu et al. [103] also consider latest-departure paths and discuss path problems and efficient algorithms to solve them. In most research conducted so far on temporal graphs, the parameters and metrics are based on the notion of temporal paths or the temporal versions of the distance, diameter, connectivity, etc., of static graphs [6, 8, 69, 81]. Note that, just like in the case of temporal paths, the notions of the temporal distance between two vertices or the temporal diameter of the graph may vary; the parameter of interest may again be the number of edges, or the duration of a temporal path, or the smallest arrival time. Kostakos [74] considers temporal graphs as a tool for temporal dataset analysis and defines metrics, such as temporal proximity and geodesic proximity. Casteigts and Flocchini [26, 27], as well as Casteigts et al. [29], also consider additional parameters such as edge latencies; all [26, 27, 29] constitute a recent attempt to integrate all the known models and concepts of dynamic graphs. The reader is also referred to the work of Michail [82] for an overview of results on temporal graph issues that have recently appeared in the Computer Science community.

The main motivation behind the study of temporal graphs lies on the areas of communication networks, complex networks and distributed computing [21, 25–27, 29, 88, 97, 98, 104], but there have also been authors, e.g. Erlebach et al. [42] and Michail and Spirakis [83], that studied temporal graphs from a purely graph theoretic aspect. The theoretical analysis of temporal graphs can provide a deeper understanding of the structural and algorithmic aspects of temporal graphs. For example, Berman [20] and Kempe et al. [69] proved that the famous Menger's theorem for vertex-disjoint paths does not hold for temporal graphs, even when each edge only appears once over time; however, Mertzios et al. [81] gave a natural temporal analogue of Menger's theorem, even when the edges appear arbitrarily many times over time. Michail and Spirakis [83] also investigated a temporal analogue of the *maximum matching* problem, showing the problem to be NP-complete in vast contrast to its polynomially solvable static version. Whether the computational complexity of a static graph problem carries over to its temporal analogue depends on the choice of problem and the metric considered; for example, although graphic TSP can be approximated within a factor $(\frac{3}{2} - \varepsilon_0)$ for connected undirected graphs [55], Erlebach et al. [42] showed that even if a temporal graph is connected at every step in time, it is NP-hard to approximate temporal exploration within $O(n^{1-\varepsilon})$, for any $\varepsilon > 0$. In [83], however, the authors show that temporal exploration can be approximated with ratio d, where d is the dynamic diameter of the temporal graph.

On another note, one may not be given a temporal graph but may be required to *design* one on a given underlying graph, usually with the requirement that the final temporal graph satisfies some property. Mertzios et al. [81] considered as cost measures of the temporal graph the *temporality* and the *temporal cost*: the former is the maximum number of times an edge becomes available and the latter is the total number of labels[1] assigned to the edges of the graph. Given a (static) graph G, it is APX-hard to compute a minimum cost assignment L that maintains all reachabilities of G and has lifetime at most equal to the diameter of G [81].

When faced with problems that are hard to solve, or to even approximate, in worst case scenarios, it may be worth looking into special cases of temporal graphs derived from some specific application domain or whose edges appear in some particular pattern. So, studying temporal graphs with specific structural characteristics may often be a natural and beneficial approach to the efficient solution of temporal graph problems. Casteigts et al. [29] establish a hierarchy of temporal graph classes based on deterministic patterns and properties such as global connectivity at every step in time. Chaintreau et al. [30] investigate a probabilistic pattern of edge availabilities, namely the case where each edge appears at each time step with some probability p. Clementi et al. [32] generalized this probabilistic model to the *edge-Markovian* case, in which the availabilities of an edge in two consecutive steps are dependent. Avin et al. [15] consider yet another probabilistic pattern, where there is a family of (static) graph instances $G_1, G_2, \ldots$ and at every time step, if the current instance is G_i, then in the next time step the instance will be G_j with some probability p_{ij}.

1.2 Summary of our Results

In this section, we provide a short overview of the problems we considered and the results we derived.

[1]Recall that the *labels* of an edge are the discrete moments in time at which the edge becomes available.

1.2.1 Ephemeral Graphs with Random Link Availabilities

As mentioned earlier, a good way to get an insight into the behaviour of temporal graphs with respect to some parameter(s) is to study an average case scenario, or a case where the edges of the underlying graph appear following some pattern. The model that we consider here is one where each edge of the underlying graph independently receives one (or more) labels chosen uniformly at random from a "pool" of available labels. The results presented are a combination of our work published in [4] and [8], and are displayed in full in Chapter 2. The general model we consider is one of *ephemeral* temporal graphs, in the sense that their lifetime is finite. The available time span, up to the lifetime of the graph, is split into equisized periods and each edge receives one or more labels chosen uniformly at random from within each period. We say that such graphs are *sparse* if the ratio of the availability instances per period over the length of the period is $o(1)$. We also define a temporal analogue of the static graph diameter, the *Temporal Diameter*, and based on it, we define *fast, efficient,* and *slow* graphs. We partially characterise fast graphs with length of period equal to the number of nodes and a single random label per period per edge. We then define the critical availability, ρ^*, as a measure of periodic random availability of the edges needed to make the graph fast. Finally, we provide a lower bound as well as an upper bound on ρ^*.

1.2.2 Temporal network flows

Network flows have been investigated before in the context of dynamic networks, where the "dynamicity" of the network lies in the existence of transit times on the edges of the network. However, not much is known for cases where the dynamicity of the network is translated as the potential of change in the topology of the underlying graph over time. Our results on temporal network flows are presented in Chapter 3. There, we introduce temporal flows on temporal networks and show that the problem of finding the maximum amount of flow that can pass from a source vertex s to a sink vertex t up to a given time is solvable in Polynomial time, even when node buffers are bounded. We then examine mainly the case of unbounded node buffers. We provide a (simplified) static *Time-Extended network*, which is of *polynomial size to the input* and whose static flow rates are equivalent to the respective temporal flow of the temporal network; via that, we prove that the maximum temporal flow is equal to the minimum *temporal s-t cut*. We further show that temporal flows can always be decomposed into flows, each of which moves only through a journey, i.e., a directed path whose successive edges have strictly increasing moments of existence. We partially characterise networks with random edge availabilities that tend to eliminate the $s \to t$ temporal flow. Finally, we consider *mixed* temporal networks, which have some edges with specified availabilities and some edges with random availabilities; we show that it is **#P**-hard to compute the *tails and expectations of the maximum temporal flow* (which is now a random variable) in a mixed temporal network.

1.2.3 Temporal Graph Design

In all our previous work, it is assumed that the temporal graph is given. However, this may not always be the case. There are applications where we may need to design a temporal graph that satisfies some property, given an underlying graph. The work described here is concerned with the property TC, which stands for *Temporal Connectivity.* A preliminary version of this work was published in [6], and an extension of it [5] is currently in the second stage of reviews for publication in the Theory of Computing Systems Journal. The details and full analysis can be found in Chapter 4. We say that a temporal graph is temporally connected when there is a journey from every vertex to every other vertex in the graph. Our first result is a polynomial-time algorithm that answers whether a given temporal graph satisfies the property TC. Following that, we mainly consider undirected graphs and derive a polynomial, asymptotically optimal w.r.t. the total number of labels used, procedure that designs a temporally connected graph on a given underlying graph. The above procedure is in fact optimal for trees; to show that, we first prove a lower bound on the number of labels of any labelling that satisfies TC on a tree and show that the suggested procedure assigns the same number of labels as the optimal for trees. However, there are pragmatic cases, where one is not free to design a temporal graph from scratch, but is instead *given* a temporal graph with the claim that it satisfies TC. In such cases, a designer's job may be to remove as many labels as possible without destroying temporal connectivity; we have shown that this problem is APX-hard, i.e., it admits no PTAS unless P=NP. On the positive side, we show that in dense graphs with random edge availabilities, a very large number of labels can asymptotically almost surely be removed without losing TC. However, a temporal graph may be *minimal*, i.e., temporal connectivity exists but is destroyed by the removal of *any* label of the graph. We show the existence of minimal temporal graphs with at least $n \log n$ labels, where n is the number of vertices in the graph.

1.2.4 Interval Temporal Graphs

This part of our work was published in [3] and is presented in full in Chapter 5. The model here is slightly different from the one studied in the previous chapters: we consider continuous time, allow edges to appear for continuous-time intervals and permit very high-speed (instantaneous) information dissemination. This is a natural assumption, since time is indeed continuous, and is motivated by several application domains such as proximity networks or infrastructural systems. We examine connectivity issues of this type of interval temporal graphs; we provide a polynomial-time algorithm that answers if a given such graph is connected during a given time period. For the case where the graph as a whole does not remain connected, we also give a polynomial-time algorithm that returns large components that remain connected (and large) during the period in question. The algorithm also considers components that start as large at the beginning of the period but disconnect into smaller ones within the period; it answers how long after the beginning of the period these components stay connected and large. Finally, using interval temporal graphs, we model a network with edges whose *lifetimes* are not controlled by us. However, we are allowed to “feed” the network with extra edges that may reconnect it when a failure happens, so that its connectivity is maintained throughout a time period. We show that we can whp maintain the connectivity for a long time by using a randomised approach. This approach also saves us some cost in the design of availabilities of the edges; here, the cost is defined as the sum, over all extra edges, of the length of their availability-to-reconnect interval.

1.3 Thesis Outline

The material covered in this thesis has been arranged in the following way:

Chapter 2

This chapter covers problems on ephemeral temporal graphs with (periodic) random availabilities of edges. We introduce a special case of ephemeral random temporal graphs, where there is a single period of available labels in the set $\{1, 2, \ldots, n\}$, where n is the number of vertices and present results obtained in [4]. We then proceed to present the results produced in [8] as an obvious path forward following the promising work accomplished in [4].

Chapter 3

Presented here are our results on temporal network flows. Our research related to this topic is a work in progress at the time of the write-up of this thesis [7]. We consider the general temporal graph model for which we show polynomial-time algorithms to solve the Maximum Temporal Flow problem. We also consider the ephemeral random temporal network model and a generalization of it, the mixed temporal network model; for them, we show that it is #**P**-hard to compute the *tails and expectations of the maximum temporal flow*.

Chapter 4

This chapter contains the details of the material presented in [6] that discusses the complexity of optimal design of temporally connected graphs. Here, we work both on the assumption that we are given an already labelled temporal graph and the assumption that we are free to "create" a temporal graph on an underlying static graph, i.e., decide the labelling of the edges of the given graph.

Chapter 5

The work analysed in this chapter was produced in [3] and focuses on verifying and maintaining connectivity of interval temporal graphs. Interval temporal graphs are different from the (traditional) temporal graph model in the sense that time is no longer considered to be discrete, but also in the sense that information dissemination and connectivity notions are defined differently.

Chapter 6

The final chapter summarises the results of this work and looks into further avenues forward for this research.

Appendix A

The appendix contains information on fundamental concepts of Computer Science that are related to the topic of this thesis or used in the main chapters of this thesis. It discusses the concept of algorithms and graph theory, as well as some probability theory and linear programming basics, and provides definitions to notions used throughout the thesis that are not only related to temporal graphs, but are broadly and generally used in Computer Science.

Each of the Chapters 2-5 concludes with a set of open problems and/or possible further research questions that may be of interest to the reader.

1.4 Author's Publications included in this Thesis

TABLE 1.1: The author's publications and co-authors throughout the duration of her Ph.D. studies.

Title	*Authors*	*Appeared*
Ephemeral Networks with Random Availability of Links: Diameter and Connectivity [4]	E.C. Akrida, L. Gąsieniec, G.B. Mertzios, and P.G. Spirakis	SPAA 2014[2]
Ephemeral Networks with Random Availability of Links: The case of fast networks [8]	E.C. Akrida, L. Gąsieniec, G.B. Mertzios, and P.G. Spirakis	JPDC 2016[3]
On Temporally Connected Graphs of Small Cost. [6]	E.C. Akrida, L. Gąsieniec, G.B. Mertzios, and P.G. Spirakis	WAOA 2015[4]
On verifying and maintaining connectivity of interval temporal networks [3]	E.C. Akrida, and P.G. Spirakis	ALGOSENSORS 2015[5]
The complexity of optimal design of temporally connected graphs [5]	E.C. Akrida, L. Gąsieniec, G.B. Mertzios, and P.G. Spirakis	TOCS (under review)[6]

[2]SPAA 2014: 26th ACM Symposium on Parallelism in Algorithms and Architectures, 2014

[3]JPDC 2016: Journal of Parallel and Distributed Computing, Volume 87, Pages 109–120, 2016

[4]WAOA 2015: 13th Workshop on Approximation and Online Algorithms, 2015

[5]ALGOSENSORS 2015: 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, 2015

[6]TOCS: Theory of Computing Systems (second stage of reviews)

Chapter 2

Random Temporal Graphs and the Temporal Diameter

This chapter is heavily based on results published in [4] at the 26th ACM Symposium on Parallelism in Algorithms and Architectures in 2014 (SPAA'14), as well as the extension of that work published in the Journal of Parallel and Distributed Computing in 2016 [8]. We consider here a model of temporal graphs, the links of which are available only at certain moments in time, chosen randomly from a subset of the positive integers. We define the notion of the Temporal Diameter of such graphs. We also define *fast* and *slow* such temporal graphs with respect to the expected value of their temporal diameter. We then provide a partial characterisation of fast random temporal graphs. We also define the critical availability as a measure of periodic random availability of the edges of a graph, required to make the graph fast. We finally give a lower bound as well as an upper bound on the (critical) availability.

2.1 Overview

2.1.1 Graphs with (periodic) random availabilities

Driven from the foundational work of Kempe et al. [69], as well as the work of Mertzios et al. [81], we consider time to be discrete, that is, we consider graphs in which the edges are available *only at certain moments in time*, e.g., days or hours.

Such graphs can be described via an underlying (di)graph $G = (V, E)$ (*the links of which is what can become available over time*) and an assignment L assigning a set of discrete labels of availability to each edge (resp. arc) of G.

We consider here both the single-label-per-edge model of [69] and the multi-labelled one, which allows links to be available at multiple times (i.e., more than one label per edge). The values assigned to each edge of the underlying graph G, i.e., the time labels of the edge, indicate the times at which we can cross it (from one end to the other in arbitrary direction, if the edge is undirected, or from its start to its end, if the edge is directed).

In the context of this chapter, we mainly study *random temporal graphs*, in which the labels assigned to the edges of the underlying graph are chosen at random from a set of available time labels. In particular, we focus here on temporal graphs, the labels of the edges of which are integers randomly chosen by the following procedure:

Label Assignment Procedure(LAP)

Let k, α, ρ be positive integers greater than or equal to 1, with $\rho \leq \alpha$. Let N_L be the set $N_L = \{1, 2, \ldots, L\}$, where $L = \alpha k$.

(I) N_L is partitioned into k consecutive *periods* $\Pi_1, \Pi_2, \ldots, \Pi_k$, each of α consecutive integers.

(II) For every edge (arc), e, of $G = (V, E)$ and every given period Π_j, we draw ρ integers independently and uniformly at random from the set $\{(j-1)\alpha + 1, (j-1)\alpha + 2, \ldots, j\alpha\}$ and assign the set of those integers (omitting duplicates), denoted by $L(j, e)$, to e. The label set of the edge e, L_e, is the union of $L(j, e)$ over all periods.

We call ρ the *availability* of labels per period, or simply the (period) availability. We call $\frac{\rho}{\alpha}$ the *density* of the periodic random availabilities. We call α the *length of the period.*

Definition 2.1 (Ephemeral random (temporal) graph). Let $G = (V, E)$ (resp. $D = (V, E)$) be a graph (resp. digraph) where V is the set of vertices and E is the set of edges (resp. arcs). Let each edge (resp. arc) be assigned labels (availability instances) by the Label Assignment Procedure, with k periods, $\Pi_i,\ i = 1, \ldots, k$, length of any period equal to $\alpha \geq 1$ and density $\frac{\rho}{\alpha}$. We call the

resulting graph an *ephemeral random temporal graph* $N(k, \alpha, \rho)$.
If $\alpha = n$, then the graph is called a *normalised ephemeral random temporal graph*, $N(k, n, \rho)$.
The number $T_{max} = \alpha k$ is called the (maximum) *lifetime* of $N(k, \alpha, \rho)$.

Note that such graphs are indeed *ephemeral*: no edge is available after T_{max}.

The set L_e of labels of any edge (arc) e is the set of exactly all the moments in time at which e is available (for use). One might associate the periods to, say, "months" and the sets $L(j, e)$ (for period Π_j and edge e) as the sets of (randomly chosen) days in which e is available.

In many situations, availability of links comes at a cost. Available links may correspond, e.g., to connections in physical systems requiring high energy. They may also correspond to very rare moments in time, in which the link of a hostile network is "unguarded" and, thus, one can pass a message at that time without putting the message in danger.

2.1.2 Journeys, the Temporal Diameter and Sparsity

Recall the definitions of a time-edge, a (foremost) journey, and the temporal distance between two vertices:

Definition 2.2 (**Time edge**). Let $e = \{u, v\}$ (resp. $e = (u, v)$) be an edge (resp. arc) of the underlying (di)graph of a temporal graph and consider a label $l \in L_e$. The ordered triplet (u, v, l) is called *time edge*.

Definition 2.3 (**Journey**). A *temporal path* or *journey* j from a vertex u to a vertex v ($u \to v$ *journey*) is a sequence of time edges (u, u_1, l_1), (u_1, u_2, l_2), … , (u_{k-1}, v, l_k), such that $l_i < l_{i+1}$, for each $1 \leq i \leq k - 1$. We call the last time label of journey j, l_k, *a*rrival time of the journey.

Definition 2.4 (**Foremost journey**). A $u \to v$ journey j in a temporal graph is called *foremost journey* if its arrival time is the minimum arrival time of all $u \to v$ journeys' arrival times, under the labels assigned to the underlying graph's edges.

Definition 2.5 (**Temporal distance**). The *temporal distance* of a (target) vertex v from a (source) vertex u is the arrival time of the foremost $u \to v$ journey and is denoted by $\delta(u, v)$.

We now define the Temporal Diameter of a temporal graph as the maximum temporal distance:

Definition 2.6 (Temporal diameter). Let $G(L)$ be a temporal graph. The *temporal diameter* of $G(L)$, denoted by TD, is the maximum, over all ordered pairs of vertices $s, t,\ s \neq t$, $\delta(s,t)$.

Note that, for a given labelling L, even for a connected graph G, there may be two vertices s and t so that there is no journey from s to t. In this case, the temporal diameter is $TD(G(L)) = +\infty$

Definition 2.7 (Expected temporal diameter). Consider instances $G(L)$ of an ephemeral random temporal graph $N(k, \alpha, \rho)$. Each instance corresponds to a labelling drawn by the random choices. The expected temporal diameter of N is the expectation $E[TD]$ over all such instances. We denote it by $ETD(N)$.

Note that, for any $N(k, \alpha, \rho)$, over a connected graph G, with k being at least equal to the diameter of G, the $ETD(N)$ is *finite* and it is at most $T_{max} = \alpha k$, since every edge from a vertex s to a vertex t, $s, t \in V(G)$, will get at least one label of availability in each period. That is, for any connected graph $G = (V, E)$, a number of periods $k \geq diam(G)$, where $diam(G)$ is the diameter, suffices to bound the temporal distance between any two vertices $u, v \in V(G)$. This holds even when $\rho = 1$, and even when the single label chosen per period in this case is selected arbitrarily. So, we get the following remark:

Remark. Let G be a connected graph (resp. a strongly connected digraph). Let $diam(G)$ be the diameter of G. Consider the ephemeral random temporal graph $N(k, \alpha, \rho)$ with $k \geq diam(G)$ and $\alpha, \rho \geq 1$. Then, in any instance $I = G(L)$ of it and for any pair of vertices s, t, it holds that $\delta_I(s,t) \leq \alpha \cdot diam(G)$, where $\delta_I(s,t)$ is the temporal distance of t from s in I. Thus, $ETD(N) \leq \alpha \cdot diam(G)$.

In this Chapter, we consider *sparse* ephemeral random temporal graphs (and also the case of a single period) and wish to characterize *fast* random temporal graphs.

Definition 2.8 (Sparse). An ephemeral random temporal graph $N(k, \alpha, \rho)$ is called *sparse* if $\frac{\rho}{\alpha} \in o(1)$.

Definition 2.9 (Fast). We call a normalised ephemeral random temporal graph $N(k, n, \rho)$ *fast* if $ETD(N) = O(\log n)$.

Clearly, fast graphs need just one period for all-to-all journeys to exist with high probability. A major goal of this Chapter is to characterize fast and sparse ephemeral random temporal graphs.

Definition 2.10 (Efficient). An ephemeral (normalised) random temporal graph $N(k, \alpha, \rho)$ is called *efficient* if $ETD(N) = o(\alpha \cdot diam(G))$, where G is the underlying connected graph (resp. strongly connected digraph).

Definition 2.11 (Slow). An ephemeral (normalised) random temporal graph $N(k, \alpha, \rho)$ is called *slow* if $ETD(N) = \Theta(\alpha \cdot diam(G))$, where G is the underlying connected graph (resp. strongly connected digraph).

To further motivate some of the questions raised in this work, consider a very *hostile* clique, G, the edges of which are usually guarded. Whenever an edge is guarded it is impossible to pass a message through it. We may pass a message to a neighbour in G only when the link to this neighbour is unguarded (i.e., available). Now, let us assume that each edge will become available only at a single *random moment* in every period and also $k = 1$, i.e., only one period exists. Let us examine the normalised case, where $\alpha = n$. After time n, no link of the clique is ever available. Such a (random) time of availability indicates a break in the security of the link. How fast can we pass a message (starting from a vertex s) to all the other vertices in the clique? Certainly, one possibility is to wait, for each destination t, for the link (s, t) to become available. But this may mean a passing time equal to $\frac{n}{2}$ in expectation. Can we spread a message faster? We show here that for the temporal clique with a single random moment of availability per link, one can still pass the message to all vertices in time $\Theta(\log n)$ with high probability and on the average. This means that a sparse normalised random temporal clique is *fast*. That is, a seemingly very hostile clique (each link of which is unguarded only for one random moment) is, in fact, not so secure with respect to fast dissemination of enemy information.

2.1.3 Previous work

2.1.3.1 Relation to the Random Phone-Call Model

The first logarithmic time results for probabilistic information dissemination were obtained in the classical Random Phone-Call model defined by Demers et al. [34].

In [34], the authors present a push algorithm that uses $\Theta(\log n)$ time and $\Theta(n \log n)$ message transmissions. For complete graphs of size n, Frieze and Grimmett [52] presented an algorithm that broadcasts in time $\log_2 n + \ln n + o(\log n)$ with a probability of $1 - o(1)$. Later, Pittel [93] showed that (with probability $1 - o(1)$) it is possible to broadcast a message in time $\log_2 n + \ln n + f(n)$, where $f(n)$ can be any slow growing function.

Karp et al. [67] presented a push and pull algorithm which reduces the total number of transmissions to $O(n \log \log n)$, with probability $1 - n^{-1}$, and showed that this result is asymptotically optimal. For sparser graphs it is not possible to stay within $O(n \log \log n)$ message transmissions together with a broadcast time of $O(\log n)$ in this phone-call model, not even for random graphs [40]. However, if each node is allowed to remember a small number of neighbours to which it has communicated in some previous steps, then the number of message transmissions can be reduced to $O(n \log \log n)$, with probability $1 - n^{-1}$ [19, 41].

The network model adopted in this chapter resembles the Random Phone-Call model to some extent, however, it is essentially different. The dependence of the temporal diameter on the lifetime, for example, cannot be captured by the random phone-call model. The model described here is, in fact, considerably weaker. In the phone-call model, each node, at each step, can communicate with a random neighbour (in fact, a node may do this at several times). In our model, each link is given some (maybe even a single) random moments of existence, by the input. A node can send via this link only at that moment. That is, random availability of links is not a part of our algorithmic techniques and can not be used at arbitrary time steps.

2.1.3.2 Other related work

In this section we provide a short survey of papers with studies on networks labelled by time units or segments.

Labelled Graphs. Labelled graphs have been widely used both in Computer Science and in Mathematics, e.g., [87].

Single-labelled and multi-labelled Temporal Networks. The model of temporal graphs that we consider in this Chapter is a direct extension of the single-labelled model studied in [69] as well as the multi-labelled model studied in [81].

The prior results of [69, 81] do not consider randomness at all, and therefore are different in nature to this work. The initial paper [69] considers the case of one (non-random) label per edge and examines shortest journey algorithms. The second paper [81] extends this (non-random) model to many labels per edge and mainly examines the number of labels needed to guarantee several graph properties with certainty.

Continuous Availabilities (Intervals). Some authors have assumed the availability of an edge for a whole time-interval $[t_1, t_2]$ or multiple such time-intervals and not just for discrete moments as we assume here. Examples of such studies are [3, 25, 46].

Dynamic Distributed Networks. In recent years, there is a growing interest in distributed computing systems that are inherently dynamic. Some examples of recent work on the topic include the following: Angluin et al. [12], as well as Michail et al. [84, 85], study models of interactions among members of a distributed population of finite-state agents and discuss fundamental questions about their computational power. Avin et al. [15] study the expected cover time of simple random walks on dynamic undirected graphs with fixed underlying vertex set, under the assumption that the graph is being modified by an oblivious adversary. Casteigts et al. [29] present a hierarchical classification of time-varying graphs, where each class corresponds to a significant property examined in the distributed computing literature. Kuhn et al. [75] investigate distributed computation in dynamic networks, such as how to determine the size of the network or compute any computable function of the initial inputs. Michail et al. [86] study the propagation of influence and computation in dynamic distributed systems. Various other authors [23, 28, 32, 37, 89, 95] have also studied communication issues in dynamic distributed systems.

Distance labelling. A distance labelling of a graph G is an assignment of unique labels to vertices of G so that the distance between any two vertices can be inferred from their labels alone [53, 68].

Random labellings. Random temporal networks have been considered before, e.g., in [30, 32]. Chaintreau et al. [30] model opportunistic mobile networks as a type of random temporal networks, where each edge exists at each time-step with a fixed probability, and show a small diameter in general for that type of networks. Clementi et al. [32] examine the speed of information dissemination in a type of dynamic graphs, where each edge exists at each time-step with some probability depending on whether it existed in the previous time-step.

2.1.4 Our contribution

We introduce the model of random ephemeral temporal graphs $N(k, \alpha, \rho)$ of k periods of availability of edges and ρ moments of availability per period, chosen uniformly at random from a set of α available labels per period. We define *sparse, fast, slow* and *efficient* such graphs with respect to their expected Temporal Diameter, which is also defined here. We give a partial characterisation of the fast graphs of the form $N(k, n, 1)$. Namely, they include the class of temporal graphs on instances, $D_{n,p}$, of the directed Erdös-Renyi graphs, with probability of existence of a directed edge equal to $p \geq \varepsilon$, for some constant ε independent of the size of the underlying graph. We also give an example of a slow graph, namely an ephemeral random temporal graph on the line graph. We define the critical availability, ρ^*, of randomly available instances of an edge during a period so that the resulting graph is fast. We show that ρ^* can be unbounded, using the example of the star graph where we cannot bound the critical availability from above by any constant. Finally, we present a general (non-constant) upper bound on ρ^*.

2.2 Sparse and fast ephemeral random temporal graphs

2.2.1 The case of an underlying Erdös-Renyi random graph

Definition 2.12 (Erdös-Renyi graphs). An instance of $G_{n,p}$ (resp. $D_{n,p}$) is formed when for every pair of vertices (resp. ordered pair of vertices) u, v among a total number of n vertices, the edge $\{u, v\}$ (resp. the arc (u, v)) is chosen to exist with probability p independently of any other edge (resp. arc).

Let us consider the case where the underlying graph is an instance of the Erdös-Renyi $G_{n,p}$ ($D_{n,p}$ for digraphs) with $p \geq \varepsilon$, where ε is a constant independent of n. We will show the following:

Theorem 2.13. *Consider sparse random normalised temporal graphs $N(k, n, \rho)$, $k \geq n-1$, $1 \leq \rho \leq c$, where $c \geq 1$ is an integer independent of n. Let the underlying graph be any instance of the $D_{n,p}$ model, where $p \geq \varepsilon$ ($\varepsilon > 0$ a constant). Let A be the event that the instance, i.e., the underlying graph, is strongly connected. Then, conditioning on A, the conditional expectation of the temporal diameter is $E[TD(N)|A] = O(\log n)$, i.e., all such graphs are fast.*

Proof. Clearly, if N_1 is such a graph and N_2 is defined on the same $D_{n,p}$ but with $\rho = c = 1$, then $ETD(N_1) \leq ETD(N_2)$, since the increased availability of each edge per period may only introduce better journeys, i.e., journeys with smaller arrival time, to the graph. So, we fix $\rho = c = 1$. We will first consider the case, where the total number of periods is $k = 1$, and show that, for any two particular vertices s and t, it holds that:

$$Pr[\delta(s,t) = \Theta(\log n)] \geq 1 - \frac{1}{n^4}$$

The result for the expectation of the maximum value of $\delta(s,t)$ will follow, by noticing that in this case, $\delta(s,t) \leq n \cdot diam(D)$, where D is any strongly connected instance of $D_{n,p}$, and thus $\delta(s,t) \leq n^2$. The probability space of the examined case is the joint space S obtained by two independent experiments:

(a) Draw an instance D of $D_{n,p}$.

(b) Assign a single label chosen uniformly at random from $\{1, 2, \ldots, n\}$ to each arc of D independently of the assignments to the other arcs.

We first show the following lemma:

Lemma 2.14. *For any two particular vertices s, t, $s \neq t$, and for the probability space S, it holds:*

$$Pr[\delta(s,t) \leq \gamma \ln n] \geq 1 - \frac{3}{n^5}, \textit{ for some constant } \gamma > 1$$

Proof. We will use the method of *deferred decisions* to prove our result. This means that when we first examine a possible arc (u, v), the probability that the arc exists is p and the probability that it is assigned a particular label l, given its existence, is $Pr[(u, v)$ is assigned label $l/(u, v)$ exists$] = \frac{1}{n}$. So, if Δ is any sub-set

$\{i, i+1, \ldots, j\},\ j > i$ of $\{1, 2, \ldots, n\}$ and $\mathcal{E}$ is the event that a message can be sent via (u, v) in a specific random instance in Δ, then we have:

$$\begin{aligned} Pr[\mathcal{E}] &= Pr[(u,v) \text{ exists}] \cdot Pr[\text{the label of } (u,v) \text{ is in } \Delta/(u,v) \text{ exists}] \\ &= \frac{\Delta}{n} p \end{aligned} \qquad (2.1)$$

We will use relation (2.1) repeatedly below. We now fix p to be $p = \varepsilon$, for some constant $0 < \varepsilon < 1$. If p is larger, then the possibility of existence of every arc increases, thus the probability $Pr[\delta(s,t) \leq a]$ increases, for all a and all ordered pairs of vertices s, t which are connected via a (directed) path in the underlying digraph.

We will provide an algorithmic construction, namely Algorithm 1, which constructs a journey with logarithmic arrival time from any given vertex s to any given vertex t, with high probability. Let $d \in \Theta(\log n)$ and let $\gamma_i,\ i = 1, 2$, be suitable constants. We will only consider labels which belong in the following sequence of consecutive intervals (all of which are in the first period):

$$\begin{aligned} \Delta_1 &= (0, \gamma_1 \ln n] \\ \Delta_2 &= (\gamma_1 \ln n, \gamma_1 \ln n + \gamma_2] \\ \Delta_3 &= (\gamma_1 \ln n + \gamma_2, \gamma_1 \ln n + 2\gamma_2] \\ &\ldots \\ \Delta_{d+1} &= (\gamma_1 \ln n + (d-1)\gamma_2, \gamma_1 \ln n + d\gamma_2] \\ \Delta^* &= (\gamma_1 \ln n + d\gamma_2, 2\gamma_1 \ln n + d\gamma_2] \\ \Delta'_{d+1} &= (2\gamma_1 \ln n + d\gamma_2, 2\gamma_1 \ln n + (d+1)\gamma_2] \\ &\ldots \\ \Delta'_2 &= (2\gamma_1 \ln n + (2d-1)\gamma_2, 2\gamma_1 \ln n + 2d\gamma_2] \\ \Delta'_1 &= (2\gamma_1 \ln n + 2d\gamma_2, 3\gamma_1 \ln n + 2d\gamma_2] \end{aligned}$$

Note that the length of the total time interval we consider, described by the span of the union of all the above intervals, is $\Theta(\log n)$. Also, note that any directed path $(s, v_1, v_2, \ldots, v_{d+1}, v^*, v'_{d+1}, \ldots, v'_2, v'_1, t)$ with consecutive labels -one per edge- $\lambda_i \in \Delta_i, \lambda^* \in \Delta^*, \lambda'_i \in \Delta'_i,\ i = 1, 2, \ldots, d+1$, is a journey and its arrival time is

$\Theta(\log n)$. Algorithm 1 demonstrates the algorithmic construction of journeys like that. In the algorithm, the label assigned to the arc (u, v) is denoted by l_{uv}.

```
Algorithm 1: The Expansion Process algorithm
Input: An instance, $N(1, n, 1)$, of an ephemeral sparse random temporal graph,
       the underlying digraph, $D = (V, E)$, of which is an instance of $D_{n,\varepsilon}$, and
       vertices $s, t,\ s \neq t$
Output: An $s \to t$ journey with arrival time $\Theta(\log n)$, if such a journey exists.
$\Gamma_1(s) = \{v \in V : l_{sv} \in \Delta_1\}$;
for $i = 2$ to $d + 1$ do
    $\Gamma_i(s) = \{v \in V : \exists w \in \Gamma_{i-1}(s)$ such that $l_{wv} \in \Delta_i\}$ ;
end for
$\Gamma'_1(t) = \{v \in V : l_{vt} \in \Delta'_1\}$;
for $i = 2$ to $d + 1$ do
    $\Gamma'_i(t) = \{v \in V : \exists w \in \Gamma'_{i-1}(s)$ such that $l_{vw} \in \Delta'_i\}$;
end for
if $\exists u \in \Gamma_{d+1}(s), v \in \Gamma'_{d+1}(t)$ such that $l_{uv} \in \Delta^*$ then
    A journey from $s$ to $t$ has been created on the concatenation of the directed
    path from $s$ to $u$, the arc $(u, v)$ and the directed path from $v$ to $t$;
    return success and the journey;
else
    return failure;
end if
```

Note. The construction of a fast journey as demonstrated in Algorithm 1 may also fail if any of the $\Gamma_i, \Gamma'_i,\ i = 1, 2, \ldots, d + 1$, as defined within the algorithm, is empty.

We proceed to calculate the probability of success of Algorithm 1. We will denote by p_1 the probability that a particular arc out of s exists and is assigned a label in Δ_1. We have:

$$\begin{aligned} p_1 &= \varepsilon \cdot \frac{\gamma_1 \ln n}{n} \\ &= \frac{c_1 \ln n}{n}, \text{ where } c_1 = \varepsilon\gamma_1. \end{aligned} \tag{2.2}$$

In fact, p_1 is also the probability that a particular arc (u, v), with $u \in \Gamma_{d+1}(s), v \in \Gamma'_{d+1}(t)$, exists and is assigned a label in Δ^*.

In addition, denote by p_2 the probability that a vertex v in $\Gamma_{i-1}(s),\ i \geq 2$ (resp. a v in $\Gamma'_{i-1}(t),\ i \geq 2$) has a particular outgoing arc (resp. incoming arc) with a label in the interval Δ_i (resp. Δ'_i). We have:

$$\begin{aligned} p_2 &= \varepsilon \cdot \frac{\gamma_2}{n} \\ &= \frac{c_2}{n}, \text{ where } c_2 = \varepsilon\gamma_2. \end{aligned} \tag{2.3}$$

Note. In the following analysis, we reveal each possible arc and the arc's random label only once, when examined (*delayed revelation of random values*). Thus, we are consistent with the fact that the input is a specific instance.

Algorithm 1 describes a (limited) expansion process. Figure 2.1 illustrates how the expansion process from s to t works. That is, starting from s, we find the set $\Gamma_1(s)$ of vertices to which there is an edge from s with label in Δ_1, then the set $\Gamma_2(s)$ of vertices to which there is an edge from a vertex in $\Gamma_1(s)$ with label in Δ_2, etc. We show that, with high probability, there is a journey from s to t through vertices in the consecutive Γ_i sets.

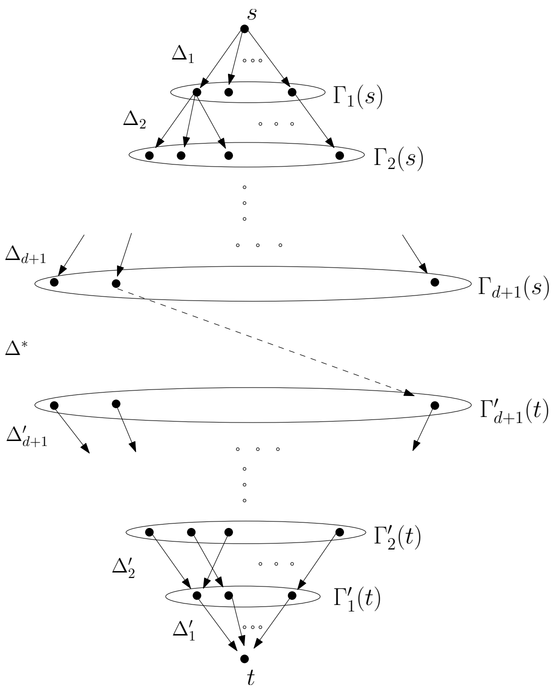


Figure 2.1: The Expansion Process.

(a) **The first step of the expansion process.**

The first step of the expansion process aims to establish with high probability a number of $\Theta(\log n)$ neighbours of s, so that the label from s to each one of them is in Δ_1. Note that the probability that an arc $(s, u), u \in V$, exists and has a label in Δ_1 is exactly $p_1 = \frac{c_1 \ln n}{n}$.

Let $\mathcal{E}_1$ be the event that $\frac{1}{2}E[|\Gamma_1(s)|] \leq |\Gamma_1(s)| \leq \frac{3}{2}E[|\Gamma_1(s)|]$.

Lemma 2.15. *It holds that:*

$$Pr[\mathcal{E}_1] = Pr\left[|\Gamma_1(s)| \in \left(\frac{1}{2}, \frac{3}{2}\right)\left(c_1 \ln n(1-\frac{1}{n})\right)\right] \geq 1 - \frac{2}{n^6}.$$

Proof. Note that:

$$E[|\Gamma_1(s)|] = (n-1)p_1 = (n-1)\frac{c_1 \ln n}{n}.$$

By the Chernoff bound (see Appendix A) on the Binomial $B(N_0, p_1)$, where $N_0 = n-1$, $\forall \delta \in (0,1)$, we have:

$$Pr[\#successes \in (1 \pm \delta)N_0 p_1] \geq 1 - 2e^{-\frac{\delta^2}{3}N_0 p_1}.$$

Now, use $\delta = \frac{1}{2}$. We get:

$$\begin{aligned} Pr[\#successes \in (\frac{1}{2}, \frac{3}{2})N_0 p_1] &\geq 1 - 2e^{-\frac{1}{12}N_0 p_1} \\ &\geq 1 - 2e^{-\frac{1}{12}(c_1 \ln n - \frac{c_1 \ln n}{n})} \\ &\geq 1 - 2e^{-\frac{1}{12}(c_1 - 1)\ln n} \\ &= 1 - \frac{2}{n^{\frac{c_1 - 1}{12}}}. \end{aligned}$$

We can choose $c_1 \geq 73$, and thus have $\frac{c_1-1}{12} \geq 6$. This completes the proof of Lemma 2.15. ■

(b) **The expansion phase until reaching $\Theta(\sqrt{n})$ vertices via journeys.**

We now show that given:

- $|\Gamma_1(s)| \in \Theta(\log n)$, and
- the probability that a potential edge e exists and has a label in a particular interval $\Delta_i, i = 2, \ldots, d+1$, is exactly $p_2 = \varepsilon \cdot \frac{|\Delta_i|}{n} = \frac{c_2}{n}$,

the vertices reachable from s via temporal paths grow (almost) geometrically. In particular, let us now condition on the event that $\frac{1}{8}c_1 \ln n \leq |\Gamma_i(s)| \leq \lambda\sqrt{n}$, for some fixed $\lambda > 0$. To find the set $\Gamma_{i+1}(s)$, we consider the vertices which are *not* in all the $\Gamma_j(s)$, $j = 1, 2, \ldots, i$ (and the fact that we look for directed edges), i.e.,

$$n_i = n - |\bigcup_{j=1}^{i} \Gamma_j(s)|.$$

The probability that a vertex u (out of the n_i vertices) belongs to $\Gamma_{i+1}(s)$ is exactly the probability that some arc $(v, u), v \in \Gamma_i(s)$, exists and has a label in the interval Δ_{i+1}, i.e., equal to:

$$\begin{aligned} q &= 1 - Pr[u \notin \Gamma_{i+1}(s)] \\ &= 1 - (1 - p_2)^{|\Gamma_i(s)|} \\ &= 1 - (1 - \frac{c_2}{n})^{|\Gamma_i(s)|}. \end{aligned}$$

We need the following fact:

Fact 1. It holds that $(1 - \frac{c_2}{n})^{|\Gamma_i(s)|} \le 1 - \frac{c_2|\Gamma_i(s)|}{2n}$.

Proof. Let $p = \frac{c_2}{n}$ and $k = |\Gamma_i(s)|$. We first need to show the following:

Lemma 2.16. *It holds that:*

$$(1 - p)^k \le 1 - kp + \binom{k}{2} p^2. \qquad (2.4)$$

Proof. We show the lemma by induction on k. For $k = 1$ the inequality holds; it actually becomes an equality for both $k = 1$ and $k = 2$. Assume now that the inequality holds for some k. Recall also that $\binom{k+1}{2} = \binom{k+1-1}{2-1} + \binom{k+1-1}{2}$. We have:

$$\begin{aligned} (1 - p)^{k+1} &= (1 - p)^k (1 - p) \\ &\le \left(1 - kp + \binom{k}{2} p^2\right)(1 - p) \\ &= 1 - kp + \binom{k}{2} p^2 - p + kp^2 - \binom{k}{2} p^3 \\ &= 1 - (k + 1)p + \left(\binom{k}{2} + \binom{k}{1}\right) p^2 - \binom{k}{2} p^3 \\ &\le 1 - (k + 1)p + \binom{k+1}{2} p^2. \end{aligned}$$

■

We will show that:

$$-kp + \binom{k}{2} p^2 \le -\frac{kp}{2}$$

and, thus, by relation 2.4 it holds that:

$$(1 - p)^k \le 1 - \frac{kp}{2}.$$

Indeed, we have:

$$\begin{aligned}
-kp + \binom{k}{2}p^2 &\le -\frac{kp}{2} \Leftrightarrow \\
\frac{k(k-1)}{2}p &\le \frac{k}{2} \Leftrightarrow \\
(k-1)p &\le 1 \Leftrightarrow \\
(|\Gamma_i(s)| - 1)c_2 &\le n.
\end{aligned}$$

The latter holds for n sufficiently large, so we have now proven Fact 1. ■

We also need the following fact:

Fact 2 (Bernoulli's inequality)**.** It holds that $(1 - \frac{c_2}{n})^{|\Gamma_i(s)|} \ge 1 - \frac{c_2|\Gamma_i(s)|}{n}$.

Therefore, we have:

$$\begin{aligned}
1 - (1 - \frac{c_2|\Gamma_i(s)|}{n}) &\ge q \ge 1 - (1 - \frac{c_2|\Gamma_i(s)|}{2n}) \Leftrightarrow \\
\frac{c_2|\Gamma_i(s)|}{n} &\ge q \ge \frac{c_2|\Gamma_i(s)|}{2n} \Leftrightarrow \\
\frac{\lambda c_2}{\sqrt{n}} &\ge q \ge \frac{c_1 c_2 \ln n}{16n} = q'.
\end{aligned} \tag{2.5}$$

The random variable $|\Gamma_{i+1}(s)|$ follows the Binomial distribution $B(n_i, q)$ and dominates $B(n_i, q')$. So, by the Chernoff bound (see Appendix A; with $\delta = \frac{1}{2}$), we have:

$$Pr\big[|\Gamma_{i+1}(s)| \in (\frac{1}{2}n_i q, \frac{3}{2}n_i q)\big] \ge 1 - 2e^{-\frac{1}{12}n_i q'}. \tag{2.6}$$

But, $n_i \ge n - (\lambda\sqrt{n})d \ge \frac{n}{2}$. So, relation 2.6 becomes:

$$\begin{aligned}
Pr\big[|\Gamma_{i+1}(s)| \in (\frac{1}{2}n_i q, \frac{3}{2}n_i q)\big] &\ge 1 - 2e^{-\frac{1}{24}n\frac{c_1 c_2 \ln n}{16n}} \\
&= 1 - 2e^{-\frac{1}{384}c_1 c_2 \ln n} \\
&= 1 - \frac{2}{n^{\frac{c_1 c_2}{384}}}.
\end{aligned}$$

We will select c_2 so that $\frac{c_1 c_2}{384} \geq 6$. So, with probability at least $1 - \frac{2}{n^6}$, we have:

$$\frac{3}{2} n_i q \geq |\Gamma_{i+1}(s)| \geq \frac{1}{2} n_i q$$

Due to relation 2.6, the latter gives:

$$\begin{aligned} \frac{3}{2} n_i \frac{c_2 |\Gamma_i(s)|}{n} &\geq |\Gamma_{i+1}(s)| \geq \frac{1}{2} n_i \frac{c_2 |\Gamma_i(s)|}{2n} \xrightarrow{1 \geq \frac{n_i}{n} \geq \frac{n}{2}} \\ \frac{3}{2} c_2 |\Gamma_i(s)| &\geq |\Gamma_{i+1}(s)| \geq \frac{1}{8} c_2 |\Gamma_i(s)|. \end{aligned}$$

We have proved that the event

$$\mathcal{E}_i = \text{“}|\Gamma_{i+1}(s)| \text{ is at most } \frac{3}{2} c_2 |\Gamma_i(s)| \text{ and at least } \frac{1}{8} c_2 |\Gamma_i(s)|\text{”}$$

holds with probability at least $1 - \frac{2}{n^6}$, provided that $\frac{1}{8} c_1 \ln n \leq \Gamma_i(s) \leq \lambda \sqrt{n}$. Thus, by conditioning on the event $\mathcal{E} = \bigcap_{i=1}^{d} \mathcal{E}_i$, we have that:

$$|\Gamma_{d+1}(s)| \geq \frac{1}{8} (\frac{c_2}{8})^d c_1 \ln n$$

and also:

$$|\Gamma_{d+1}(s)| \leq \frac{3}{2} (\frac{3c_2}{2})^d c_1 \ln n.$$

Choose d so that $\frac{3}{2}(\frac{3c_2}{2})^d c_1 \ln n \leq \xi \sqrt{n}$, for some constant $\xi > 0$, thus choose some d s.t.:

$$d \leq \frac{\ln \frac{2\xi\sqrt{n}}{3c_1 \ln n}}{\ln \frac{3c_2}{2}}$$

and also:

$$\frac{1}{8}(\frac{c_2}{8})^d c_1 \ln n > \sqrt{n} \Rightarrow d > \frac{\ln \frac{8\sqrt{n}}{c_1 \ln n}}{\ln \frac{c_2}{8}}$$

Notice that the above choice is always possible.

The probability that one or more of the events $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_d$ fail is (by the union bound) at most:

$$d \frac{2}{n^6} \leq c' \ln n \frac{2}{n^6} \leq \frac{1}{n^5}, \text{ for some } c' > 0$$

Thus, we have shown the following:

Corollary 2.17. *With probability at least* $1 - \frac{1}{n^5}$*, the expansion process out of* s *arrives at* $\Theta(\sqrt{n})$ *vertices with temporal paths of length* $d+1 \in \Theta(\log n)$*,* *consistently labelled in the intervals* Δ_i, $i = 1, 2, \dots, d+1$, *in time at most* $c_1 \ln n + dc_2 \in \Theta(\log n)$.

(c) **The reverse expansion process (out of t)**

Consider the edges reaching t *reversed* and consider the process that labels them in Δ'_1. Let $\Gamma'_1(t)$ be the vertices derived in this way, i.e., reaching t with an edge labelled in Δ'_1. Continue the reverse expansion process until we reach $\Theta(\sqrt{n})$ vertices. We consider this process, and all the random experiments within, independently of the expansion process out of s. So, we get exactly the same result as in Corollary 2.17:

Corollary 2.18. *The expansion process out of t arrives at $\Theta(\sqrt{n})$ vertices with temporal paths (reverse direction) of length $d{+}1 \in \Theta(\log n)$, consistently labelled in the intervals Δ'_i, $i = 1, 2, \dots, d+1$. Thus, it arrives to each of these vertices in time at most $c_1 \ln n + dc_2 \in \Theta(\log n)$ with probability at least $1 - \frac{1}{n^5}$.*

(d) **The matching argument**

The probability that both $|\Gamma_{d+1}(s)|$ and $|\Gamma'_{d+1}(t)|$ are of size at least $\lambda'\sqrt{n}$, $\lambda' > 0$, is at least $1 - 2\frac{1}{n^5}$. Note that we just need one arc (v_1, v_2), $v_1 \in \Gamma_{d+1}(s)$, $v_2 \in \Gamma'_{d+1}(t)$, with label in the interval Δ^* in order to demonstrate the existence of a temporal path of arrival time at most $\Theta(\log n)$ from s to t. Note also that for a *given pair of vertices* (v_1, v_2), $v_1 \in \Gamma_{d+1}(s)$, $v_2 \in \Gamma'_{d+1}(t)$, the arc appears and has a label in Δ^* with probability exactly:

$$p_1 = \varepsilon \cdot \frac{|\Delta^*|}{n} = \frac{c_1 \ln n}{n}.$$

Thus, the probability of the event A = "existence of such an edge" is:

$$p = 1 - \left(1 - \frac{c_1 \ln n}{n}\right)^{|\Gamma_{d+1}(s)| \cdot |\Gamma'_{d+1}(t)|},$$

and due to Corollaries 2.17 and 2.18, we have:

$$\begin{aligned} p &\geq 1 - \left(1 - \frac{c_1 \ln n}{n}\right)^{(\lambda')^2 n} \\ &\geq 1 - e^{-(\lambda')^2 c_1 \ln n} \\ &= 1 - \frac{1}{n^{(\lambda')^2 c_1}}. \end{aligned}$$

We can choose c_1 through the analysis so that we have:

$$p \geq 1 - \frac{1}{n^5}.$$

The probability of any of the events of Corollaries 2.17 and 2.18 or event A failing is at most $3\frac{1}{n^5}$.

This completes the proof of Lemma 2.14. ■

For the temporal graphs N considered here, namely any $N(k, n, 1)$ on instances of $D_{n,\varepsilon}$, we have in fact shown that:

$$Pr[TD(N) \leq \gamma \ln n] \geq 1 - \frac{3}{n^3}, \text{ for some constant } \gamma > 1,$$

since the probability that there exists a pair of vertices s, t for which the construction fails is at most $n^2 \frac{3}{n^5} = \frac{3}{n^3}$. We will now show the following:

Lemma 2.19. *$Pr[TD(N) \leq \gamma \ln n$, given that N is over a strongly connected instance of $D_{n,\varepsilon}] \geq 1 - \frac{4}{n^3}$, for some constant $\gamma > 1$.*

Proof. Consider the event, A', that the temporal diameter of N is $TD(N) \leq \gamma \ln n$, for some constant $\gamma > 1$, and the event, A, that the instance D of $D_{n,p}$ on which N is considered is strongly connected. As the event A' implies that the considered instance D of $D_{n,\varepsilon}$ is strongly connected, the analysis of the expansion process actually shows that $Pr[TD(N) \leq \gamma \ln n$ and D is strongly connected$] = Pr[A' \cap A] \geq 1 - \frac{3}{n^3}$.

For an instance D of $D_{n,p}$ to be strongly connected, it suffices for the undirected version of D to be connected and for any arc (u, v) that exists in D, that the arc (v, u) also exists. So,

$$Pr[A] = Pr[D \text{ of } D_{n,p} \text{ is strongly connected}] \geq Pr[G \text{ of } G_{n,p^2} \text{ is connected}].$$

The connectivity threshold of $G_{n,p}$ is $p_0 = 2\frac{\ln n}{n}$ [24]. In our case, the probability of existence of any undirected edge is $p^2 = \varepsilon^2$, which, for sufficiently large n, is greater than this threshold. Therefore:

$$Pr[A] \geq Pr[G \text{ of } G_{n,p^2} \text{ is connected}] \geq 1 - o(1).$$

So, the probability that $TD(N) \leq \gamma \ln n$, given that N is over a strongly connected instance D of $D_{n,\varepsilon}$, is:

$$\begin{aligned} Pr[A'|A] &= \frac{Pr[A' \cap A]}{Pr[A]} \\ &\geq \frac{1-\frac{3}{n^3}}{1-o(1)} \\ &\geq 1-\frac{4}{n^3}. \end{aligned}$$

This completes the proof of Lemma 2.19. ■

However, for any strongly connected instance D of $D_{n,\varepsilon}$, its diameter is at most $n-1$ and thus its Temporal Diameter is at most $n(n-1) < n^2$. So, conditioning on the event that the instance D of $D_{n,\varepsilon}$ is strongly connected, it holds that:

$$\begin{aligned} ETD(N) &\leq \gamma \ln n Pr[TD(N) \leq \gamma \ln n] + n^2(1 - Pr[TD(N) \leq \gamma \ln n]) \\ &\leq \gamma \ln n(1-\frac{4}{n^3}) + n^2\frac{4}{n^3} \\ &\leq \gamma \ln n + \Theta(\frac{1}{n}), \text{ for some constant } \gamma > 1. \end{aligned}$$

This completes the proof of Theorem 2.13. ■

Our analysis holds for any $\varepsilon > 0$ so it also holds for $\varepsilon = 1$. Thus,

Corollary 2.20. *If* $N(k, n, 1)$, $k \geq n-1$, *is defined on the directed clique then* $ETD(N) \leq \gamma \ln n$, *for some constant* $\gamma > 1$.

Note also that Lemma 2.14 and Theorem 2.13 extend trivially to the *undirected* $G_{n,\varepsilon}$ cases; an edge $\{u, v\}$ corresponds to both arcs (u, v) and (v, u) and the analysis is not significantly affected.

2.2.2 Spreading a message in random temporal graphs

Consider the following general protocol for broadcasting a message from any vertex s in the graph.

```
1 for any vertex $u \in V(G)$, $u \neq s$, and any moment $t = 1, 2, \ldots$ in time do
2     if $u$ has received the message from $s$ before $t$ and an edge (arc) out of $u$ becomes available at time $t$ then
3         Send the message through that edge (arc) at time $t$;
4     end if
5 end for
```

Clearly, this protocol spreads the message from s to any vertex, for any temporal ephemeral graph $N(k, \alpha, \rho)$ in time at most $ETD(N)$.

2.2.3 A case of slow graphs

In contrast to the fast graphs we studied in the previous Section, there are also graphs that are slow; recall that an ephemeral random temporal graph $N(k, \alpha, \rho)$ on a connected (di)graph G is said to be slow if $ETD(N) \in \Theta(\alpha \cdot diam(G))$.

Lemma 2.21. *There exists a slow ephemeral random temporal graph $N(k, n, 1)$ on a connected underlying graph G of n vertices.*

Proof. Consider the line graph G of n vertices, that is the graph which itself is a path $(e_1 = \{v_1, v_2\}, e_2 = \{v_2, v_3\}, \ldots, e_{n-1} = \{v_{n-1}, v_n\})$. Let $N(k, n, 1)$, $k \geq diam(G) = n-1$, be a random temporal graph on G and suppose that v_1 wishes to send a message to v_n in the graph. Now, let us consider the progress made within the first period, Π_1, meaning the length of the longest journey starting from v_1. Denote this progress by x_1 and let λ be an integer from 1 to $n-1$. Also, denote by l_i, $i = 1, 2, \ldots, n-1$ the label assigned to the edge e_i in the first period. It holds that:

$$\begin{aligned} Pr[x_1 \geq \lambda] &= Pr[l_1 < l_2 < \ldots < l_\lambda] \\ &= \frac{\binom{n}{\lambda}}{n^\lambda}. \end{aligned}$$

So, the expected value of the progress of the first period is:

$$\begin{aligned}
E[x_1] &= \sum_{\lambda=1}^{n-1} \lambda \cdot Pr[x_1 = \lambda] \\
&= 1 \cdot Pr[x_1 = 1] + 2 \cdot Pr[x_1 = 2] + \ldots + (n-1) \cdot Pr[x_1 = n-1] \\
&= \sum_{\lambda=1}^{n-1} Pr[x_1 \geq \lambda] \\
&= Pr[x_1 \geq 1] + \sum_{\lambda=2}^{n-1} Pr[x_1 \geq \lambda] \\
&= 1 + \sum_{\lambda=2}^{n-1} \frac{\binom{n}{\lambda}}{n^\lambda} \\
&\leq 1 + \sum_{\lambda=2}^{n-1} \frac{\frac{n^\lambda}{\lambda!}}{n^\lambda} \\
&= 1 + \sum_{\lambda=2}^{n-1} \frac{1}{\lambda!} \\
&= 1 + \frac{1}{2!} + \frac{1}{3!} + \sum_{\lambda=4}^{n-1} \frac{1}{\lambda!} \\
&\leq 1 + \frac{1}{2} + \frac{1}{6} + \sum_{\lambda=4}^{n-1} \frac{1}{2^\lambda} \\
&= 1 + \frac{1}{2} + \frac{1}{6} + \sum_{\lambda=0}^{n-1} \frac{1}{2^\lambda} - 1 - \frac{1}{2} - \frac{1}{4} - \frac{1}{8} \\
&= \frac{1 - \frac{1}{2^n}}{\frac{1}{2}} - \frac{5}{24} \\
&= 2 - \frac{5}{24} - \frac{1}{2^{n-1}} \\
&< 2.
\end{aligned}$$

Also,

$$\begin{aligned}
E[x_1] &= 1 + \sum_{\lambda=2}^{n-1} \frac{\binom{n}{\lambda}}{n^\lambda} \\
&\geq 1.
\end{aligned}$$

Therefore, the expected length of the longest journey starting from v_1 within the first period, Π_1, is more than 1 and less than 2. Let $u = v_\mu$, for some $\mu = 2, 3, \ldots, n$, be the last vertex of this journey. For the message from v_1 to continue its way towards v_n, we would then need to use labels from the next period, Π_2.

Notice that the assignment of labels for any period is independent of the assignment of labels for any other period. Therefore, the analysis for the progress, x_i, within any period Π_i, $i = 2, 3, \ldots, k$, would be the same as the analysis for the progress within Π_1. Consequently, the expected value of the progress made within each of the k periods is the same as $E[x_1]$, that is more than 1 and less than 2. So, for the message starting from v_1 to reach v_n, we would need to use a number of periods between $\frac{n-1}{2}$ and $n-1$, by linearity of expectation. Since we use at least the first $\frac{n-1}{2}$ periods and since each period has a total of n available labels, the expected arrival time of the $v_1 \to v_n$ journey under consideration will be at least $\frac{n-1}{2} \cdot n$. It will also be at most $n \cdot (n-1)$, since we will not need more than $n-1$ total periods.

Since v_1 and v_n are the two vertices with the largest distance between them in G and because of the fact that G is the line graph, it holds that $ETD(N) = E[\delta(v_1, v_2)] \in \Theta(n^2) \equiv \Theta(n \cdot diam(G))$. ■

2.3 On efficient random temporal graphs

Let $N(k, \alpha, \rho)$ be the ephemeral random temporal graph defined on an underlying undirected, connected graph $G = (V, E)$. (Similar considerations hold for strongly connected digraphs). We know that for $k \geq diam(G)$, it is $ETD(N) \leq \alpha \cdot diam(G)$. Intuitively, when we assign enough random labels per edge per period, i.e., the density $\frac{\rho}{\alpha}$ increases, then we may reduce the expected temporal diameter. One would even hope, for suitable density, that $ETD(N) \leq \alpha$, i.e., any vertex can reach any other vertex via a journey within the first period whp.

Definition 2.22 (**Critical availability, critical density**). Consider the normalised ephemeral random temporal graph $N(k, n, \rho)$ on an underlying connected graph $G = (V, E)$, with $k \geq diam(G)$. Let ρ^* be a positive integer such that:

(a) when $\rho \geq \rho^*$, then $ETD(N) \leq n$, and

(b) if $\rho = o(\rho^*)$, then $Pr[TD(N) \leq n] = o(1)$.

We call the density $\frac{\rho^*}{n}$ the *critical density* corresponding to G, and the value ρ^* the *critical availability*.

2.3.1 Graphs with ρ^* bounded below by $\log n$

We will now prove the following theorem:

Theorem 2.23. *There are connected graphs, G, (even with diameter $diam(G) = 2$) for which the critical availability is $\rho^* = \Theta(\log n)$.*

Proof. We consider the star graph of n vertices, denoted here by G_n, i.e., the tree of n vertices with one root and $n-1$ leaves. Let $N = N(k, n, \rho)$ on G_n.

(i) We first establish that $\rho(n) = \Theta(\log n)$ random labels per edge per period suffice to have $ETD(N) \leq n$. Let $\rho(n) = r \ln n$, for some $r > 1$. Denote by c the centre vertex, i.e., the root, of G_n. Now consider two fixed leafs, u_1, u_2, of G_n.

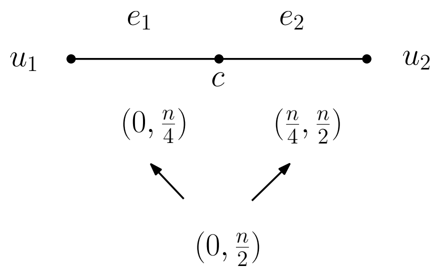


FIGURE 2.2: 2-split journey in a star graph.

Each of the edges $e_1 = \{u_1, c\}$ and $e_2 = \{c, u_2\}$ is assigned $\rho(n)$ random labels in the first period, Π_1. We call *2-split* $u_1 \to u_2$ journey any $u_1 \to u_2$ journey, where the first time edge has a label within the interval $\Delta_1 = (0, \frac{n}{4})$ and the second time edge has a label within the interval $\Delta_2 = (\frac{n}{4}, \frac{n}{2})$(see Figure 2.2). The probability that a label of e_1 falls within the interval Δ_1 is $\frac{1}{4}$. So, the probability that no label of e_1 falls within this interval is:

$$\begin{aligned} Pr[\text{no label of } e_1 \text{ is in } \Delta_1] &= \left(1 - \frac{1}{4}\right)^{r \ln n} \\ &\leq e^{-\frac{r}{4} \ln n} \\ &= \frac{1}{n^{\frac{r}{4}}}. \end{aligned}$$

Similarly, the probability that a label of e_2 falls into Δ_2 is $\frac{1}{4}$. So, the probability that no label of e_2 falls within this interval is:

$$Pr[\text{no label of } e_2 \text{ is in } \Delta_2] = \left(1 - \frac{1}{4}\right)^{r \ln n}$$

$$\leq \frac{1}{n^{\frac{r}{4}}}.$$

Now, let E be the event that some label of e_1 is in Δ_1 *and* some label of e_2 is in Δ_2. Then, by independence of label assignments:

$$Pr[E] = (1 - Pr[\text{no label of } e_1 \text{ is in } \Delta_1]) \cdot (1 - Pr[\text{no label of } e_2 \text{ is in } \Delta_2]).$$

So, we have:

$$\begin{aligned} Pr[E] &\geq \left(1 - \frac{1}{n^{\frac{r}{4}}}\right)^2 \\ &\geq 1 - \frac{2}{n^{\frac{r}{4}}}. \end{aligned}$$

Therefore, for $\frac{r}{4} > 4$, i.e., $r > 16$, we have:

$$Pr[\exists s, t \in V(G_n),\ s \neq t : \nexists 2-split\ s \to t\ journey] \leq n(n-1)\frac{2}{n^{\frac{r}{4}}} < \frac{2}{n^2}.$$

But, for $r > 16$, it also holds that:

$$\begin{aligned} Pr[TD(N) \leq \frac{n}{2}] &\geq Pr[\forall s, t \in V(G_n),\ s \neq t, \exists s \to t\ journey] \\ &\geq 1 - \frac{2}{n^2}. \end{aligned}$$

Therefore,

$$\begin{aligned} ETD(N) &\leq \frac{n}{2}(1 - \frac{2}{n^2}) + 2n\frac{2}{n^2} \\ &\leq \frac{n}{2} + \frac{3}{n} \\ &< n. \end{aligned}$$

(ii) We now prove condition (b) of Definition 2.22 for the star graph. In particular, we show that if $\rho = o(\log n)$, then $Pr[TD(N) \leq n] \leq \frac{1}{n}$. Since we look for the TD to be at most n, let us henceforward consider only the first period, Π_1. Suppose that, through an assignment L, each edge of G_n now receives $k = \frac{\ln n}{\beta(n)}$ random labels (from the set $\{1, 2, \ldots, n\}$), where $\beta(n) \to +\infty$ as $n \to +\infty$. Consider two fixed leafs $u_1, u_2 \in V(G_n)$ and let $e_1 = \{u_1, c\}, e_2 = \{c, u_2\}$ and E_{u_1,u_2} be the following event:

$$E_{u_1,u_2} = \text{“}\textit{There exists no } u_1 \to u_2 \textit{ journey in } (G_n, L)\text{”} \equiv$$

$$\equiv \text{“}\exists a \in \{2,3,\ldots,n-2\} : \textit{ all of } e_1\textit{'s labels fall within } [a,n] \textit{ and all of } e_2\textit{'s labels within}(0,a]\text{”}.$$

Given a specific $a \in \{2,3,\ldots,n-2\}$, the probability that all of e_1's labels fall within $[a,n]$ *and* all of e_2's labels fall within $(0,a]$ is:

$$\begin{aligned} &Pr\big[\text{all of } e_1\text{'s labels fall within } [a,n] \\ &\qquad \text{and all of } e_2\text{'s labels fall within } (0,a]\big] \\ &= (1-\frac{a-1}{n})^k(\frac{a}{n})^k. \end{aligned}$$

Now, the probability that event E_{u_1,u_2} occurs is at least as large as the probability that all of e_1's labels fall within $[a,n]$ *and* all of e_2's labels fall within $(0,a]$, for a specific $a \in \{2,3,\ldots,n-2\}$, e.g., for $a=\frac{n}{2}$. So:

$$\begin{aligned} Pr[E_{u_1,u_2}] &\geq Pr\big[e_1\text{'s labels fall within } [\frac{n}{2},n] \text{ and} \\ &\qquad e_2\text{'s labels fall within } (0,\frac{n}{2}]\big] \\ &= (\frac{1}{2}+\frac{1}{n})^k(\frac{1}{2})^k \\ &\geq (\frac{1}{2})^k(\frac{1}{2})^k = (\frac{1}{2})^{2k} = \frac{1}{2^{2k}}. \end{aligned}$$

The probability that no a exists, such that all of e_1's labels fall within $[a,n]$ *and* all of e_2's labels fall within $(0,a]$, is:

$$Pr[\neg E_{u_1,u_2}] = 1 - Pr[E_{u_1,u_2}] \leq 1 - \frac{1}{2^{2k}}.$$

Note that also $Pr[E_{u_2,u_1}] \geq \frac{1}{2^{2k}}$ (by symmetry).

In the star graph G_n, we can group the leafs in $\lfloor \frac{n-1}{2} \rfloor = n'$ disjoint pairs $\{u_1,u_2\},\{u_3,u_4\},\ldots,\{u_{2n'-1},u_{2n'}\}$ defining the paths $(start, centre, end)$ $P_1=(u_1,c,u_2), P_2=(u_3,c,u_4),\ldots,P_{n'}=(u_{2n'-1},c,u_{2n'})$. These paths receive *independent labels since no edges of P_i overlap with any edge of P_j, $i,j=1,2,\ldots,n'$, $i \neq j$.* So:

$$\begin{aligned} Pr[\neg E_{u_i,u_j} \text{ holds for all these pairs } u_i,u_j] &\leq (1-\frac{1}{2^{2k}})^{n'} \\ &\leq e^{-\frac{n'}{2^{2k}}}, \end{aligned}$$

i.e.,

$$Pr[TD(N) \leq n] \leq e^{-\frac{n'}{2^{2k}}}. \tag{2.7}$$

We will show that $\frac{n'}{2^{2k}} \geq \ln n$. Since $k = \frac{\ln n}{\beta(n)}$, we need to show the following:

$$\begin{aligned} \frac{n'}{2^{2\frac{\ln n}{\beta(n)}}} &\geq \ln n \Leftrightarrow \\ \lfloor\frac{n-1}{2}\rfloor &\geq 4^{\frac{\ln n}{\beta(n)}} \ln n \end{aligned} \tag{2.8}$$

It holds that:

$$\begin{aligned} 4^{\frac{\ln n}{\beta(n)}} \ln n &= n^{\frac{\ln 4}{\beta(n)}} \ln n \\ &\leq n^{\frac{2}{\beta(n)}} \ln n \\ &\leq n^{\frac{1}{2}} \ln n, \text{ since } \beta(n) > 4. \end{aligned}$$

Now, for n large enough, it also holds that:

$$n^{\frac{1}{2}} \ln n \leq \frac{n-1}{3},$$

and that:

$$\frac{n-1}{3} \leq \lfloor\frac{n-1}{2}\rfloor.$$

This proves Equation 2.8.

Therefore, it holds that $\frac{n'}{2^{2k}} \geq \ln n$ and, thus, we get from Equation 2.7 that:

$$Pr[TD(N) \leq n] \leq e^{-\ln n} = \frac{1}{n}.$$

■

2.3.2 An upper bound on ρ^*

Consider $N(k, n, \rho)$ on a connected undirected $G = (V, E)$, where $|V| = n$ and $k \geq diam(G)$.

For each edge e of G, consider a structure $s(e)$ being a sequence of boxes $B_1(e), B_2(e), \ldots, B_{diam(G)}(e)$ (see Figure 2.3).

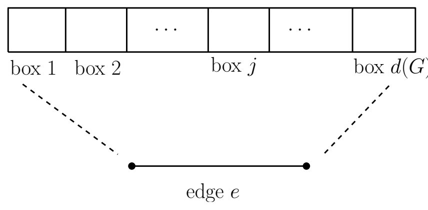


FIGURE 2.3: Structure $s(e)$ of consecutive ranges of labels.

Let each Box_i of e be assigned to a corresponding range (sequence) $L_i(e)$ of labels, each of size (*#labels*) equal to $\lambda = \frac{n}{diam(G)}$, so that:

$$\forall i = 1, 2, \ldots, diam(G),$$
$$Box_i \text{ corresponds to } L_i(e) = \{(i-1)\lambda + 1, \ldots, i\lambda\}$$

Claim 1. If for all $e \in E(G)$ and for all $Box_i(e)$, at least one label of $L_i(e)$ gets into $Box_i(e)$, then $TD(N) \leq n$, i.e., any vertex can reach any other vertex via a journey within the first period.

Proof. For any s, t, any shortest path p *from* s *to* t will be of length $|p| \leq diam(G)$. Any edge e may be at any "position" in p (first, second, ..., last) or not belong to p at all. The journey from s to t is the path $p = \big(e_{p_1} = \{s, u_1\}, e_{p_2} = \{u_1, u_2\}, \ldots, e_{p_{last}} = \{u_{|p|-1}, t\}\big)$ which, for each edge e_{p_i}, uses a label that is in the box $Box_i(e_{p_i})$. ■

Note now that when we assign a random label to edge e for the first period, Π_1, the probability that this label falls in $Box_i(e)$ is exactly $\frac{\lambda}{n}$. For ρ random labels assigned to e for the first period, Π_1, the probability that none of them falls in $Box_i(e)$ is $\left(1 - \frac{\lambda}{n}\right)^{\rho}$. Thus, the probability of the event:

$$A(e) = \text{"there exists a box of } e \text{ without a label in the first period"}$$

is at most $diam(G)\left(1 - \frac{\lambda}{n}\right)^{\rho}$.

Clearly,

$$\left(1 - \frac{\lambda}{n}\right)^{\rho} \leq e^{-\frac{\lambda\rho}{n}} = e^{-\frac{\rho}{diam(G)}}$$

and since $diam(G) \leq n$, it is enough to have $\frac{\rho}{diam(G)} \geq 4 \ln n$ to get $diam(G)\left(1 - \frac{\lambda}{n}\right)^{\rho} \leq n\frac{1}{n^4} = \frac{1}{n^3}$.

Then, the probability of the event:

$$A = \text{“there exists an edge } e \in E(G) \text{ such that } A(e) \text{ holds”}$$

is at most $n^2 \frac{1}{n^3} = \frac{1}{n}$.

But,

$$\frac{\rho}{diam(G)} \geq 4 \ln n \Leftrightarrow \rho \geq 4\ diam(G) \ln n.$$

Indeed, if we assign at least 4 $diam(G) \ln n$ labels per edge per period, we get:

$$\begin{aligned} ETD(N) &\leq n(1 - \frac{1}{n}) + n^2 \frac{1}{n} \\ &\leq n - 1 + \frac{1}{n} \\ &\leq n. \end{aligned}$$

Therefore, it must be $\rho^* \leq 4\ diam(G) \ln n$.

2.4 Open Problems

In this work, we extend the research on temporal graphs, by introducing and studying the model of random ephemeral temporal graphs. A further goal of our research is to fully characterize the fast such graphs, in which information dissemination can be expected to be very quick. To do the latter, we need to show that expansion processes similar to the one presented in this Chapter reach most of the graph quickly, and to find the smallest probability for which the expansion process (and, thus, the information dissemination in the graph) is fast.

We also aim in establishing a tighter upper bound on the critical availability. One could try to express the Expected Temporal Diameter as a function that depends not only on n but also on ρ, and try to minimise it with respect to the latter.

Yet another goal for our future research is to study models of random temporal graphs, where the random selection of availability labels for the edges of the underlying graph follows distributions, other than the uniform.

Chapter 3

Temporal Network Flows

Network flows is a fascinating topic that combines what many researchers and specialists like best about mathematics and computer science; it fuses a complex mathematical subject and an enormous range of applications in a wide range of sciences and fields, such as computer science, physics, chemistry, social systems, transportation and many others. This Chapter introduces temporal flows in temporal networks. The work analysed here has not been published yet at the moment of the write-up of this thesis. The results presented are heavily based on work conducted in cooperation with Jurek Czyzowicz, Leszek Gąsieniec, Łukasz Kuszner, and Paul Spirakis. We consider a new model of temporal networks, called *mixed* temporal networks. These networks have some edges with known, pre-defined labels and some edges that receive labels drawn independently and uniformly at random from the set $\{1, \ldots, \alpha\}$, for some $\alpha \in \mathbb{Z}^+$. Clearly, traditional temporal networks are a special case of mixed temporal networks, having only edges with pre-defined, specific labels. Also, the ephemeral random temporal networks (with a single period) of Chapter 2 can be considered a special case of mixed temporal networks, where all edges receive their labels at random. In this Chapter, we introduce the notions of temporal flow networks and temporal flows and we study the complexity of computing the maximum amount of flow that can pass from a source vertex to a sink vertex up to a given moment in time.

3.1 Overview

3.1.1 Our model and the problem

Consider a temporal network on an underlying directed graph $G(V, E)$ with a set V of n vertices (nodes) and a set E of m edges (links). Let $s, t \in V$ be two special vertices of G, called the *source* and the *sink*, respectively. For simplicity, let us assume that no edge enters the source s and no edge leaves the sink t. We also assume that an infinite amount of a quantity, say, a liquid, is available in s at time zero. The liquid that is initially located at node s can flow in the network through edges only at days at which the edges are available. Each edge $e \in E$ in the network is also equipped with a *capacity* $c_e > 0$ which is a positive integer, unless otherwise specified. We also consider each node $v \in V$ to have an internal buffer (storage) $B(v)$ of maximum size B_v; here, B_v is also a positive integer; initially, we shall consider both the case where $B_v = +\infty$, for all $v \in V$, and the case where all nodes have finite buffers. From Section 3.3 on, we only consider unbounded (infinite) buffers.

The *semantics* of the flow of our liquid within G are the following:

- Let an amount x_v of liquid be at node v, i.e., in $B(v)$, at the *beginning* of day l, for some $l \in \mathbb{N}$. Let $e = (v, w)$ be an edge that exists at day l. Then, v may *push* some of the amount x_v through e at day l, as long as that amount is at most c_e. This quantity will arrive to w at the *end* of *the same day*, l, and will be stored in $B(w)$.
- At the end of day l, for any node w, some flow amounts may arrive from edges (v, w) that were available at day l. Since each such quantity of liquid has to be stored in w, the sum of all flows incoming to w plus the amount of liquid that is already in w at the end of day l, after w has sent any flow out of it at the beginning of day l, must not exceed B_w.
- Flow arriving at w at (the end of) day l can leave w only via edges existing at days $l' > l$.

Thus, our flows are not flow rates, but flow amounts (similar to considerations in *transshipment problems*).

Notice that we assume above that we have absolute knowledge of the days of existence of each edge. This information is detailed, but it can model a range of scenarios where a network is operated by many users and detailed description of link existence (or lack thereof) is needed; for example, one may need to have detailed information on planned maintenance of pipe-sections in a water network to assure restoration of the network services, or one may need to know in advance the time schedule of a rail network to circulate passengers. However, such a detailed input can not be used in all practical cases; often, instead of having a specific list of days of existence of some edge(s), one may be able to obtain statistical knowledge of a pattern of existence of connections via previously gathered information. A model that captures such cases is the model of *Mixed Temporal Networks*, which we introduce and study here, along with the traditional Temporal Networks model.

We provide efficient solutions to the *Maximum temporal flow problem* (MTF): Given a directed graph G with edge availabilities, distinguished nodes s, t, edge capacities and node buffers as previously described, and also given a specific day $l' > 0$, find the maximum value of the quantity of liquid that can arrive to t by (the end of) day l'.

Notice that no flow will arrive to t in fewer days than the "temporal distance of t from s" (the smallest *arrival time* of any $s \to t$ journey).

MTF is very different from the problem of standard (instantaneous) flows. Indeed, in the network of Figure 3.1 with all node buffers and edge capacities being infinite, but *all edges existing only at the same day*, say $l = 5$, no flow can *ever* arrive to t. Moreover, MTF has not been a well examined problem in previous work conducted

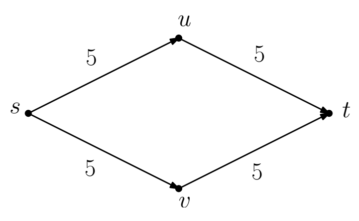


FIGURE 3.1: Difference between temporal flows and standard flows.

on (continuous or discrete) *dynamic flows* [18, 46, 56, 65, 73]. Indeed, the "transit time" on each edge of our networks is less than one day, and *only* if the edge exists at that day. *All* units of flow that are located at the tail of an edge at a moment when the edge becomes available may pass through the edge all together (like a "packet" of information), if the edge capacity allows it. Also, in our model, the existence of node buffers (holdover flow) is *necessary*; *in contrast to all previous flow studies*, our networks cannot propagate flow without holdover flows, i.e., node buffers storing flow units.

So, we consider here ephemeral networks that change over time, as well as flows that are dynamic and the movement of which is determined by the temporal structure of the network.

3.1.2 Previous work

The traditional (static) network flows were extensively studied in the seminal book of Ford and Fulkerson [48] (see also Ahuja et al. [2]) and the relevant literature is vast. *Dynamic network flows* (see, e.g., [64]) refer to *static* directed networks, the edges of which have capacities as well as transit times. Ford and Fulkerson [48] formulated and solved the dynamic maximum flow problem. For excellent surveys on dynamic network flows, the reader is also referred to the work of Aronson [13], the work of Powell [94], and the great survey by Skutella [96]. Dynamic network flows are also called *flows over time*. In [73], the authors review *continuous* flows over time where $f_e(\theta)$ is the rate of flow (per time unit) entering edge e at time θ; the values of $f_e(\theta)$ are assumed to be Lebesgue-measurable functions. In our model, we assume that any flow amount that can pass through an edge at an instant of existence, will pass, i.e., our $f_e(\theta)$ is infinite in a sense. In a technical report [59], the authors examine earliest arrival flows with time-dependent travel times and edge capacities; they describe the flow equations of their model and give their own Ford-Fulkerson approach and dynamic cut definitions; although different to their model, our work gives an intuitively simpler definition of a temporal cut and an approach to a temporal version of the Ford-Fulkerson algorithm that is easier to explain. For various problems on flows over time, see [18, 45, 54, 56, 65, 66, 72, 73]. Flows over time have been also considered in problems of scheduling jobs in a network [22].

Classical static flows have recently been re-examined for the purpose of approximating their maximum value or improving their time complexity [10, 17, 39, 58, 76, 80, 90]. Network flows have also been used in multi-line addressing [38].

As far as we know, this is the first work to examine flows on temporal networks. Berman [20] proposed a similar model to temporal networks, called *scheduled networks*, in which each edge has separate departure and arrival times; he showed that the max-flow min-cut theorem holds in scheduled networks, when edges have *unit capacities*. Recall that there is also literature on models of temporal networks with random edge availabilities [8, 30, 32], but to the best of our knowledge, this is the first work on flows in such temporal networks.

Perhaps the closest model in the flows literature to the one we consider here is the "*Dynamic*[1] dynamic network flows", studied by Hoppe in his Ph.D. thesis [64, Chapter 8]. In [64, Chapter 8], Hoppe introduces *mortal edges* that exist between a start and an end time; still, Hoppe assumes transmission rates on the edges and the ability to hold any amount of flow on a node (infinite node buffers). Thus, our model is an extreme case of the latter, since we assume that edges exist only at specific days (instants) and that our transit rates are virtually unbounded, since at one instant *any amount* of flow can be sent through an edge if the capacity allows.

3.1.3 Our contribution

We introduce flows in Temporal Networks for the first time. We are interested in the maximum total amount of flow that can pass from s to t during the lifetime of the network; notice that the edges of the network exist only at some days during the lifetime, different in general for each edge.

In Section 3.1.4, we formulate the problem of computing the maximum temporal flow and in Section 3.2, we show that it can be solved in polynomial time, even when the node capacities are finite. This is in contrast to the NP-hardness result conjectured by Hoppe [64, personal communication with Klinz] for bounded holdover flows in dynamic dynamic networks.

The remainder of the Chapter mainly concerns networks in which the nodes have unbounded buffers, i.e., buffers with infinite capacity. In Section 3.3.2, we define the corresponding time-extended network (TEG) which converts our problem to a static flow problem (following the time-extended network tradition in the literature [48]). However, we manage to simplify TEG into a *simplified time-extended*

[1]The first "dynamic" term refers to the dynamic nature of the underlying graph, i.e., appearance and disappearance of its edges

network (STEG), the size of which, i.e., number of nodes and edges, is *polynomial on the input*, and not exponential as usual in flows over time. Using the STEG, we prove our *maximum temporal flow-minimum temporal cut* theorem; temporal cuts extend the traditional cut notion, since the edges included in a cut need not exist at the same day(s) in time. We also show that temporal flows are always decomposable into a set of flows, each moving through a particular journey.

In Section 3.4.1, we demonstrate cases of graphs with random edge availabilities in which the flow that arrives at t is eliminated asymptotically almost surely.

Admittedly, the encoding of the input in our temporal network problems is quite detailed but as previously mentioned, specific description of the edge availabilities may be required in a range of network infrastructure settings where there is a planned schedule of link existence. On the positive side, some problems that are weakly NP-hard in similar dynamic flow models become polynomially solvable in our model. However, in many practical scenarios it is reasonable to assume that not all edge availabilities are known in advance, e.g., in a water network where there may be unplanned disruptions at one or more pipe sections; in these cases, one may have statistical information on the pattern of link availabilities. Here, we introduce and study flows in mixed temporal networks for the first time; these are networks in which the availabilities of some edges are random and the availabilities of some other edges are specified. In such networks, the value of the maximum temporal flow is a random variable. Consider, for example, the temporal flow network of Figure 3.2 where there are n directed disjoint two-edge paths from s to t. Assume that *every* edge independently selects a *unique* label uniformly at random from the set $\{1, \dots, \alpha\}$, $\alpha \in \mathbb{N}^*$. The edge capacities are the numbers drawn in the boxes, with $w'_i \geq w_i$ for all i. Here, the value of the maximum $s \to t$ flow is a random variable that is the sum of Bernoulli random variables. This already indicates that the exact calculation of the maximum flow in mixed networks is a hard problem. In Section 3.4.2 we show for mixed networks that it is **#P**-hard to compute tails and expectations of the maximum temporal flow.

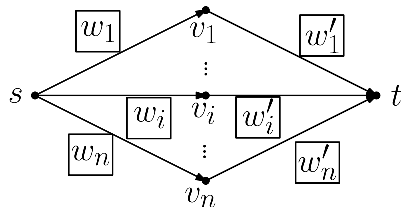


Figure 3.2: A mixed temporal network.

3.1.4 Formal Definitions

A basic assumption that we follow here is that when a (flow) entity passes through an available edge e at time t, then it can pass through a subsequent edge only at some time $t' \geq t+1$ and only at a time at which that edge is available; so, flow amounts have to follow journeys from s to t. In the tradition of assigning "transit times" in the dynamic flows literature, one may translate the use of strictly-increasing time-labels as each edge $e \in E$ having a *transit time*, tt_e, with $0 < tt_e < 1$, but *otherwise arbitrary and not specified.* Henceforth, we will use $tt_e = 0.5$ for all edges e, without loss of generality in our results; any value of tt_e between 0 and 1 will lead to the same results in this Chapter.

Definition 3.1 (Temporal Flow Network)**.** A temporal flow network $(G(L), s, t, c, B)$ is a temporal graph $G(L) = (V, E, L)$ equipped with:

1. a source vertex s and a sink (target) vertex t
2. for each edge e, a capacity $c_e > 0$; usually the capacities are assumed to be integers.
3. for each node v, a buffer $B(v)$ of storage capacity $B_v > 0$; B_s and B_t are assumed to be infinite.

If all node capacities are infinite, we denote the temporal flow network by $(G(L), s, t, c)$.

Definition 3.2 (Temporal Flows in Temporal Flow Networks)**.** Let $(G(L) = (V, E, L), s, t, c, B)$ be a temporal flow network. Let:

$$\begin{aligned} \delta_u^+ &= \{e \in E | \exists w \in V, e = (u, w)\} \\ \delta_u^- &= \{e \in E | \exists w \in V, e = (w, u)\} \end{aligned}$$

be the outgoing and incoming edges to u. Also, let $L_R(u)$ be the set of labels on all edges incident to u along with an extra label 0 (artificial label for initialization), i.e.,

$$L_R(u) = \bigcup_{e \in \delta_u^+ \cup \delta_u^-} L_e \cup \{0\}$$

A temporal flow on $G(L)$ consists of a non-negative real number $f(e, l)$ for each time-edge (e, l), and real numbers $b_u^-(l), b_u^\mu(l), b_u^+(l)$ for each node $u \in V$ and each "day" l in $L_R(u)$. These numbers must satisfy all of the following:

1. $0 \leq f(e, l) \leq c_e$, for every time edge (e, l),
2. $0 \leq b_u^-(l) \leq B_u$, $0 \leq b_u^\mu(l) \leq B_u$, $0 \leq b_u^+(l) \leq B_u$, for every node u and every $l \in L_R(u)$
3. for every $e \in E$, $f(e, 0) = 0$,
4. for every $v \in V \setminus \{s\}$, $b_v^-(0) = b_v^\mu(0) = b_v^+(0) = 0$,
5. for every $e \in E$ and $l \notin L_e$, $f(e, l) = 0$,
6. at time 0 there is an infinite amount of flow "units" available at the source node s,
7. for every $v \in V \setminus \{s\}$ and for every $l \in L$, $b_v^-(l) = b_v^+(l_{prev})$, where l_{prev} is the largest label in $L_R(v)$ that is smaller than l,
8. (Flow out on day l) for every $v \in V \setminus \{s\}$ and for every l, $b_v^\mu(l) = b_v^-(l) - \sum_{e \in \delta_v^+} f(e, l)$,
9. (Flow in on day l) for every $v \in V \setminus \{s\}$ and for every l, $b_v^+(l) = b_v^\mu(l) + \sum_{e \in \delta_v^-} f(e, l)$.

Note. One may think of $b_v^-(l), b_v^\mu(l), b_v^+(l)$ as the buffer content of liquid in v at the "morning","noon", i.e., after the departures of flow from v, and "evening", i.e., after the arrivals of flow to v, of day l.

Note. For a temporal flow f on an acyclic $G(L)$, if one could guess the (real) numbers $f(e, l)$ for each time-edge (e, l), then the numbers $b_v^-(l), b_v^\mu(l), b_v^+(l)$, for every $v \in V$, can be computed by a single pass of an ordering of the vertices of $G(L)$ from s to t. This can be done by following (1) through (9) from Definition 3.2 from s to t.

Definition 3.3 (Value of a Temporal Flow)**.** The value $v(f)$ of a temporal flow f is $b_t^+(l_{max})$ under f, i.e., the amount of liquid that, via f, reaches t during the lifetime of the network (l_{max} is the maximum label in L). If $b_t^+(l_{max}) > 0$ for a particular flow f, we say that f is *feasible.*

The value $v(f)$ of a temporal flow f is not necessarily equal to the total amount of flow that leaves s throughout the lifetime of the network; that is another difference of this model to traditional network flows.

Definition 3.4 (Mixed temporal networks)**.** Given a directed graph $G = (V, E)$ with a source s and a sink t in V, let $E = E_1 \cup E_2$, so that $E_1 \cap E_2 = \emptyset$, and:

1. the labels (availabilities) of edges in E_1 are specified, and
2. each of the labels of the edges in E_2 is drawn uniformly at random from the set $\{1, 2, \dots, \alpha\}$, independently of the others.

We call such a network *Mixed Temporal Network* $[1, \alpha]$ and denote it by $G(E_1, E_2, \alpha)$.

Note that (traditional) temporal networks as previously defined are a special case of the mixed temporal networks, in which $E_2 = \emptyset$. However, with some edges being available at random times, the value of a temporal flow (until time α) becomes a random variable and the study of relevant problems requires a different approach than the one needed for (traditional) temporal networks.

Problem (Maximum Temporal Flow (MTF))**.** Given a temporal flow network $(G(L), s, t, c, B)$ and a day $d \in \mathbb{N}^*$, compute the maximum $b_t^+(d)$ over all flows f in the network.

3.2 LP for the MTF problem with or without bounded buffers

In the description of the MTF problem, if d is not a label in L, it is enough to compute the maximum $b_t^+(l_m)$ over all flows, where l_m is the maximum label in L that is smaller than d. Henceforth, we assume $d = l_{max}$ unless otherwise specified; notice that the analysis does not change: if $d < l_{max}$, one can remove all time-edges with labels larger than b and solve MTF in the resulting network with new maximum label at most d.

Note also that, as previously mentioned, $b_t^+(l_{max})$ is not necessarily equal to the total outgoing flow from s during the lifetime of the network, where the lifetime is $l_{max} - l_{min}$, l_{min} being the smallest label in the network. For example, consider the network of Figure 3.3, where the labels of an edge are the numbers written next to it and its capacity is the number written inside the box; for $d = 5$, the *maximum flow by day* 5 *is* $b_t^+(5) = 8$, i.e., the flow where 5 units follow the journey $s \to v \to t$ and 3 units follow the journey $s \to u \to v \to t$; however, the total outgoing flow from s by day 5 is $10 > 8$.

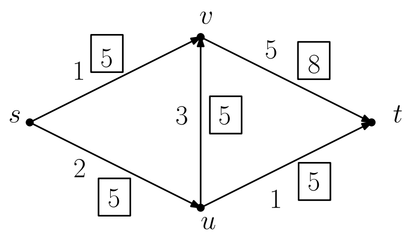


Figure 3.3: The total outgoing flow from s is not always the same as the maximum flow (at t) by some day d; here $d = 5$.

Let Σ be the set of conditions of Definition 3.2. The optimization problem, Π:

$$\left\{\begin{array}{ll} \text{max (over all } f) & b_t^+(d) \\ \text{subject to} & \Sigma \end{array}\right\}$$

is a *linear program* with unknown variables $\{f(e,l), b_v^-(l), b_v^+(l)\}$, $\forall l \in L, \forall v \in V$, since each condition in Σ is either a linear equation or a linear inequality in the unknown variables. Therefore, by noticing that the number of equations and inequalities are polynomial in the size of the input of Π, we get the following Lemma:

Lemma 3.5. *Maximum Temporal Flow is in P, i.e., can be solved in polynomial time in the size of the input, even when the node buffers are finite, i.e., bounded.*

Note. Recall that E_L is the set of time edges of a temporal graph. If $n = |V|, m = |E|$ and $k = |E_L| = \sum_e |L_e|$, then MTF can be solved in sequential time polynomial in $n+m+k$ when the capacities and buffer sizes can be represented with polynomial in n number of bits. In the remainder of the paper, we shall investigate more efficient approaches for MTF.

Note. Lemma 3.5 for bounded node buffers is in wide contrast with the claim that the corresponding problem in dynamic dynamic network flows is NP-complete [64, p. 82].

3.3 Temporal Networks with unbounded buffers at nodes

3.3.1 Basic remarks

We consider here the MTF problem for temporal networks on underlying graphs with $B_v = +\infty,\ \forall v \in V$.

Definition 3.6 (Temporal Cut)**.** Let $(G(L), s, t, c)$ be a temporal flow network on a digraph G. A set of time-edges, S, is called a temporal cut (separating s and t) if the removal from the network of S results in a temporal flow network with *no* $s \rightarrow t$ *journey.*

Definition 3.7 (Minimal Temporal Cut)**.** A set of time-edges, S, is called a minimal temporal cut (separating s and t) if:

1. it is a temporal cut, and
2. the removal from the network of any $S' \subset S$ results in a temporal flow network with at least one journey from s to t, i.e., any proper subset of S is *not* a temporal cut.

Definition 3.8. Let S be a temporal cut of $(G(L) = (V, E, L), s, t, c)$. The *capacity* of the cut is $c(S) := \sum_{(e,l)\in S} c(e, l)$, where $c(e, l) = c_e,\ \forall l$.

In Figure 3.4, the numbers next to the edges are their availability labels and the numbers in the boxes are the edge capacities; here, a minimal temporal cut is $S = \{((s, v), 1), ((s, v), 7)\}$ with capacity $c(S) = 20$. Notice that another minimal cut is $S' = \{((v, t), 8)\}$ with capacity $c(S') = 2$.

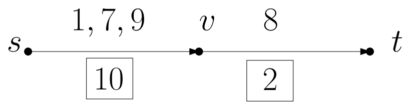


FIGURE 3.4: The set $S = \{((s, v), 1), ((s, v), 7)\}$ is a minimal temporal cut.

It follows from the definition of a temporal cut:

Lemma 3.9. *Let S be a (minimal) temporal cut in $(G(L) = (V, E, L), s, t, c)$. If we remove S from $G(L)$, no flow can ever arrive to t during the lifetime of $G(L)$.*

Proof. The removal of S leaves no $s \to t$ journey and any flow from s needs at least one journey to reach t, by definition. ■

3.3.2 The time-extended flow network and its simplification

Let $(G(L) = (V, E, L), s, t, c)$ be a temporal flow network on a directed graph G. Let E_L be the set of time edges of $G(L)$. Following the tradition in literature [48], we construct from $G(L)$ a *static* flow network called *time-extended* that corresponds to $G(L)$, denoted by $TEG(L) = (V^*, E^*)$. By construction, $TEG(L)$ admits the same maximum flow as $G(L)$. $TEG(L)$ is constructed as follows.

For every vertex $v \in V$ and for every time step $i = 0, 1, \ldots, l_{max}$, we add to V^* a copy, v_i, of v. V^* also contains a copy of v for every time edge (x, v, l) of $G(L)$; in particular, we consider a copy v_{l+tt} of v in V^*, for some $l \in \mathbb{N}$, iff $(x, v, l) \in E_L$, for some $x \in V$. Notice that $0 < tt < 1$ (by definition of the transit times), so if a vertex $v \in V$ has an incoming edge e with label l and an outgoing edge with label $l + 1$, the copies v_{l+tt}, v_{l+1} of v in V^* will never be identical (see Figure 3.5).

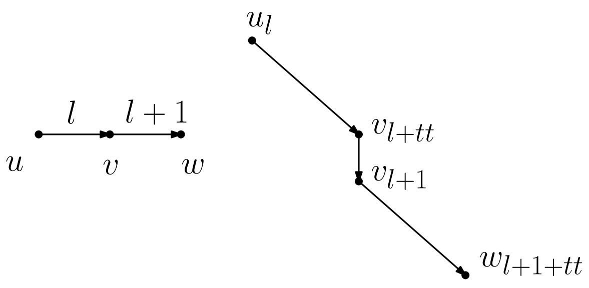


FIGURE 3.5: The copies of vertex v in the time-extended network $TEG(L)$.

E^* has a directed edge (called *vertical*) from a copy of vertex v to the *next* copy of v, for any $v \in V$. More specifically,

$$\forall v \in V,\ (v_i, v_j) \in E^* \iff \begin{cases} v_i, v_j \in V^*, & \text{and} \\ j > i, & \text{and} \\ \forall k > i : v_k \in V^* \implies k \geq j \end{cases}$$

Furthermore, for every time edge of $G(L)$, E^* has a directed edge (called *crossing*) as follows:

$$\forall u, v \in V, l \in \mathbb{N},\ (u, v, l) \in E \iff (u_l, v_{l+tt}) \in E^*$$

Every crossing edge $e \in TEG(L)$ that connects copies of vertices $u, v \in V$ has the capacity of the edge $(u, v) \in G(L)$, $c_e = c_{u,v}$. Every vertical edge $e \in TEG(L)$ has capacity $c_e = B_v = +\infty$. The source and target vertices in $TEG(L)$ are the first copy of s and the last copy of t in V^*, respectively. Note that $|V^*| \leq |V| \cdot l_{max} + |E_L|$ and $|E^*| \leq |V| \cdot l_{max} + 2|E_L|$.

We will now "simplify" $TEG(L)$ as follows: we convert vertical edges between consecutive copies of the same vertex into a *single vertical edge (with infinite capacity)* from the first to the last copy in the sequence and we remove all intermediate copies; we only perform this simplification when no intermediate node is an endpoint of a crossing edge. We call the resulting network *simplified time-extended* network and we denote it by $\text{STEG}(L) = (V', E')$.

In particular, for every vertex $v \in V$, we consider a copy v_0 of v in V'. Furthermore, we consider a copy v_l of v in V' iff there is a time edge $(v, x, l) \in E_L$, for some $x \in V$. Finally, we consider a copy v_{l+tt} of v in V' iff there is a time edge $(x, v, l) \in E_L$, for some $x \in V$.

Now, the set E' of edges in $\text{STEG}(L)$ has a directed *vertical* edge from a copy of vertex v to the *next* copy of v, for any $v \in V$. More specifically,

$$\forall v \in V,\ (v_i, v_j) \in E' \iff \begin{cases} v_i, v_j \in V', & \text{and} \\ j > i, & \text{and} \\ \forall k > i : v_k \in V' \implies k \geq j \end{cases}$$

Furthermore, for every time edge of $G(L)$, we consider the *crossing* edge as in the time-extended graph, i.e.:

$$\forall u, v \in V, l \in \mathbb{N},\ (u, v, l) \in E \iff (u_l, v_{l+tt}) \in E'$$

Every crossing edge $e \in \text{STEG}(L)$, i.e., every edge that connects copies of different vertices $u, v \in V$, has the capacity of the edge $(u, v) \in G(L)$, $c_e = c_{u,v}$. Every edge $e \in \text{STEG}(L)$ between copies of the same vertex $v \in V$ has capacity $c_e = B_v = +\infty$. The source and target vertices in $\text{STEG}(L)$ are the first copy of s and the last copy of t in V' respectively. Note that $|V'| \leq |V| + 2|E_L|$ and $|E'| \leq |V| + 3|E_L|$.

Denote the first copy of any vertex $v \in V$ in the time-extended network by v_{copy_0}, the second copy by v_{copy_1}, the third copy by v_{copy_2}, etc. Denote by δ_u^+ and δ_u^- the sets of outgoing edges from u and incoming edges to u, respectively, i.e.:

$$\begin{aligned}\delta_u^+ &= \{e \in E | \exists w \in V, e = (u, w)\} \\ \delta_u^- &= \{e \in E | \exists w \in V, e = (w, u)\}\end{aligned}$$

An $s \to t$ flow f in $G(L)$ defines an $s \to t$ flow (rate), f_R, in the time-extended network STEG(L) as follows:

- The flow from the first copy of s to the next copy is the sum of all flow units that "leave" s in $G(L)$ throughout the time the network exists:

$$f(s_{copy_0}, s_{copy_1}) := \sum_{l \in \mathbb{N}} \sum_{e \in \delta_s^+} f(e, l)$$

- The flow from the first copy of any *other* vertex to the next copy is zero:

$$\forall v \in V \setminus s,\ f(v_{copy_0}, v_{copy_1}) := 0$$

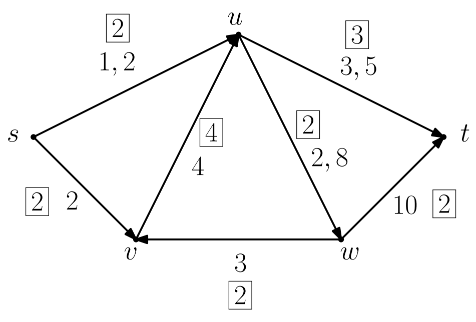
- The flow on any crossing edge that connects some copy u_l of vertex $u \in V$ and the copy v_{l+tt} of some other vertex $v \in V$ is exactly the flow on the time edge (u, v, l):

$$\forall (u_l, v_{l+tt}) \in E',\ f(u_l, v_{l+tt}) := f((u, v), l)$$

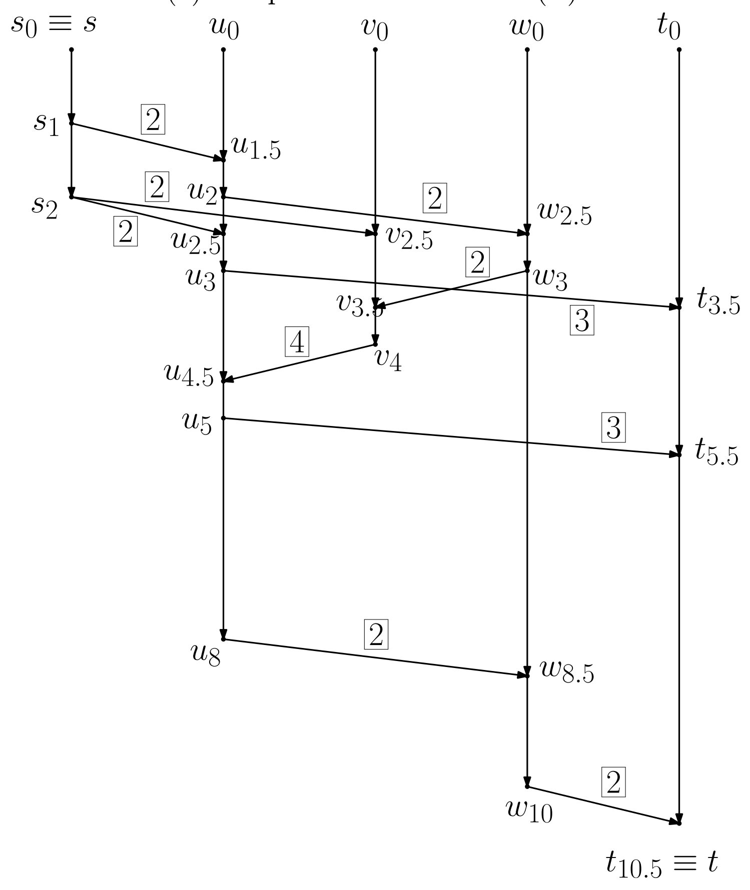
- The flow between two *consecutive* copies v_x and v_y, for some x, y, of the same vertex $v \in V$ corresponds to the units of flow stored in v from time x up to time y and is the difference between the flow *received* at the first copy through all incoming edges and the flow *sent* from the first copy through all outgoing crossing edges. So, $\forall v \in V, i = 1, 2, \ldots$, it is:

$$f(v_{copy_i}, v_{copy_{i+1}}) := \textstyle\sum_{z \in V'} f(z, v_{copy_i}) - \sum_{u \in V' \setminus v_{copy_{i+1}}} f(v_{copy_i}, u)$$

Example. Figure 3.6a shows a temporal network $G(L)$ with source s and sink t. The labels of an edge are shown next to the edge and the capacity of an edge is shown *written in a box* next to the edge. The respective simplified time-extended static graph STEG(L) is shown in Figure 3.6b. The capacity of an edge is shown *written in a box* next to the edge. Notice that edges between copies of the same vertex have infinite capacities (equal to the infinite capacity of the vertex buffer) which are not shown in the figure.



(a) Temporal flow network $G(L)$.



(b) Simplified time extended network STEG(L).

FIGURE 3.6: Constructing the simplified time-extended network.

Now, consider a temporal flow network $(G(L) = (V, E, L), s, t, c)$ and a temporal flow f in it. Let f_R be a corresponding static flow (rate) in the static network STEG(L). By the construction of STEG(L), it follows:

Lemma 3.10. *Given a temporal flow network* $(G(L) = (V, E, L), s, t, c)$ *on a directed graph* G*,*

1. *The maximum temporal flow (from s to t), $max_f v(f)$, in $G(L)$ is equal to the maximum (standard) flow rate from s to t in the* static *network* STEG(L).
2. *A temporal flow f is proper in $G(L)$ (i.e., satisfies all constraints) iff its corresponding static flow rate f_R is feasible in* STEG(L).

Lemma 3.11. *The minimum capacity s-t cut of the static network $TEG(L)$ is equal to the minimum capacity s-t cut of the static network* STEG(G).

Proof. Any minimum capacity cut in either $TEG(L)$ or STEG(L) uses crossing edges. But the crossing edges are the same in both networks. Therefore, the lemma holds. ■

We are now ready to prove the main Theorem of this section:

Theorem 3.12. *The maximum temporal flow in $(G(L) = (V, E, L), s, t, c)$ is equal to the minimum capacity (minimal) temporal cut.*

Proof. By Lemma 3.10, the maximum temporal flow in $G(L)$ is equal to the maximum flow rate from s to t in $TEG(L)$ and in STEG(L). But in STEG(L), the maximum s-t flow rate is equal to the minimum s-t cut [48]. Now, by Lemma 3.11, this cut is also equal in capacity to the minimum capacity s-t cut in $TEG(L)$. But any minimum capacity cut in $TEG(L)$ is only using crossing edges and thus corresponds to a temporal cut in $G(L)$, of the same capacity (since the removal of the respective time-edges leaves no $s \to t$ journey in $G(L)$). ■

It is also easy to see the following:

Lemma 3.13. *Any static flow rate algorithm A that computes the maximum flow in a static, directed s-t network G of n vertices and m edges in time $T(n, m)$, also computes the maximum temporal flow in a $(G(L) = (V, E, L), s, t, c)$ temporal flow network in time $T(n', m')$, where $n' \leq n + 2|E_L|$ and $m' \leq n + 3|E_L|$.*

Proof. We run A on the static network STEG(L) of n' vertices and m' edges. Note that STEG(L) is, by construction, acyclic. ■

Note. In contrast to all the dynamic flows literature, our simplified time-extended network has size (number of nodes and edges) *linear* on the input size of $G(L)$, and *not exponential.*

The following is a direct corollary of the construction of the Simplified Time-Extended Graph and shows that any temporal flow from s to t in temporal flow networks with unbounded node buffers can be decomposed into temporal flows on some $s \to t$ journeys.

Corollary 3.14 (Journeys flow decomposition). *Let $(G(L) = (V, E, L), s, t, c)$ be a temporal flow network on a directed graph G. Let f be a temporal flow in $G(L)$ (f is given by the values of $f(e, l)$ for the time-edges $(e, l) \in E_L$). Then, there is a collection of $s \to t$ journeys $j_1, j_2, \ldots, j_k$ such that:*

1. $k \leq |E_L|$
2. $v(f) = v(f_1) + \ldots v(f_k)$
3. *f_i sends positive flow only on the time-edges of j_i*

3.4 Mixed Temporal Networks

Mixed temporal networks of the form $G(E_1, E_2, \alpha)$ (see Definition 3.4) can model practical cases, where some edge availabilities are exactly specified, while some other edge availabilities are randomly chosen (due to security reasons, faults, etc.). For example, in a water network, one may be faced with scheduled, i.e., planned, disruptions for maintenance in some water pipes, but unplanned (random) disruptions in some others. With some edges being available at random times, the value of the maximum temporal flow (until time α) now becomes a random variable.

In this section, we focus our attention on temporal networks that either have all their labels chosen uniformly at random, or are (fully) mixed.

3.4.1 Temporal Networks with random availabilities that are flow cutters

We study here a special case of the mixed temporal networks $G(E_1, E_2, \alpha)$, where $E_1 = \emptyset$, i.e., *all* the edges in the network become available at random time instances. We partially characterise such networks that eliminate the flow that arrives at t.

Let $G = (V, E)$ be a directed graph of n vertices with a distinguished source, s, and a distinguished sink, t. Suppose that each edge $e \in E$ is available only at a *unique* moment in time (i.e., day) *selected uniformly at random* from the set $\{1, 2, \ldots, \alpha\}$, for some even integer $\alpha \geq 1$; suppose also that the selections of the edges' labels are independent. Let us call such a network a Temporal Network with unique random availabilities of edges, and denote it by URTN(α).

Lemma 3.15. *Let P_k be a directed $s \to t$ path of length k in G. Then, P_k becomes a journey in $URTN(\alpha)$ with probability at most $\frac{1}{k!}$.*

Proof. For a particular $s \to t$ path P_k of length k, let $\mathcal{E}$ be the event that "P_k is a journey", $\mathcal{D}$ be the event that "all k labels on P_k are different" and $\mathcal{S}$ be the event that "at least 2 out of the k labels on P_k are equal". Then, we have:

$$\begin{aligned} Pr[\mathcal{E}] &= Pr[\mathcal{E}|\mathcal{D}] \cdot Pr[\mathcal{D}] + Pr[\mathcal{E}|\mathcal{S}] \cdot Pr[\mathcal{S}] \\ &= Pr[\mathcal{E}|\mathcal{D}] \cdot Pr[\mathcal{D}] \\ &\leq Pr[\mathcal{E}|\mathcal{D}] \end{aligned}$$

Now, each particular *set* of k different labels in the edges of P_k is equiprobable. But for each such set, all permutations of the k labels are equiprobable and only one is a journey, i.e., has increasing order of labels. Therefore:

$$Pr[P_k \text{ is a journey}] \leq \tfrac{1}{k!}.$$

■

Now, consider directed graphs as described above, in which the distance from s to t is at least $c \log n$, for a constant integer $c > 2$; so any directed $s \to t$ path has at least $c \log n$ edges. Let us call such graphs " c-long $s \to t$ graphs" or simply c-long. A c-long $s \to t$ graph is called *thin* if the number of simple directed $s \to t$ paths is at most n^{β}, for some constant β.

Lemma 3.16. *Consider a $URTN(\alpha)$ with an underlying graph G being any particular c-long and thin digraph. Then, the probability that the amount of flow from s arriving at t is positive tends to zero as n tends to $+\infty$.*

Proof. The event E_1 ="at least one $s \to t$ path is a journey in $URTN(\alpha)$" is a prerequisite for a positive flow from s arriving at t. So,

$$Pr[\text{flow arriving at } t > 0] \;=\; Pr[\exists s \to t \text{ simple path in } G \text{ which is a journey}]$$

$$\begin{aligned} &\leq n^{\beta} Pr[\text{any specific simple path in } G \text{ is a journey}] \\ &\leq n^{\beta} \frac{1}{(c \log n)!}, \end{aligned} \tag{3.1}$$

by Lemma 3.15 and since every $s \to t$ path in G has length at least $c \log n$. It holds that $c! \geq \left(\frac{c}{2}\right)^{\frac{c}{2}}$ and that $(\log n)! \geq \left(\frac{\log n}{2}\right)^{\frac{\log n}{2}}$. Therefore, relation 3.4.1 becomes:

$$Pr[\text{flow arriving at } t > 0] \leq \frac{1}{\left(\frac{c}{2}\right)^{\frac{c}{2}}} \cdot \frac{n^{\beta}\sqrt{n}}{(\log n)^{\frac{\log n}{2}}}$$

But, $n^{\beta}\sqrt{n} = o\left((\log n)^{\frac{\log n}{2}}\right)$ for n large enough, so the Lemma holds. ■

3.4.2 The complexity of computing the expected maximum temporal flow

We consider here the following problem:

Problem (Expected Maximum Temporal Flow)**.** What is the time complexity of computing the *expected value* of the maximum temporal flow, v, in $G(E_1, E_2, \alpha)$?

Let us recall the definition of the class of functions **#P**:

Definition 3.17. [92, p.441] Let Q be a polynomially balanced, polynomial-time decidable binary relation. The *counting problem* associated with Q is: Given x, how many y are there such that $(x, y) \in Q$? **#P** is the class of all counting problems associated with polynomially balanced polynomial-time decidable functions.

Loosely speaking, a problem is said to be **#P**-hard if a polynomial-time algorithm for it implies that **#P** = **FP**, where **FP** is the set of functions from $\{0,1\}^*$ to $\{0,1\}^*$ computable by a deterministic polynomial-time Turing machine[2]. For a more formal definition, see [92].

We now show the following:

Lemma 3.18. *Given an integer $C > 0$, it is **#P**-hard to compute the probability that the maximum flow value v in $G(E_1, E_2, \alpha)$ is at most C, $Pr[v \leq C]$.*

[2] $\{0,1\}^* = \cup_{n \geq 0}\{0,1\}^n$, where $\{0,1\}^n$ is the set of all strings (of bits $0,1$) of length n

Proof. Recall that if $J = \{w_1, \ldots, w_n\}$ is a set of n positive integer weights and we are given an integer $C \geq \sum_{i=1}^{n} \frac{w_i}{2}$, then the problem of computing the number, T, of subsets of J with total weight at most C is **#P**-hard, because it is equivalent to counting the number of feasible solutions of the corresponding KNAPSACK instance [92].

Consider now the temporal flow network of Figure 3.7 where there are n directed disjoint two-edge paths from s to t. For the path with edges e_i, e'_i, via vertex v_i, the capacity of e_i is w_i and the capacity of e'_i is $w'_i \geq w_i$. In this network, $E_1 = \emptyset$ and $E_2 = E$, i.e., the availabilities of every edge are chosen independently and uniformly at random from $\{1, \ldots, \alpha\}$. Also, assume that each edge selects a *single* random label.

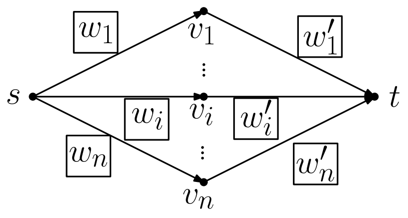


FIGURE 3.7: The mixed temporal network we consider.

Clearly, the value of the maximum temporal flow from s to t until time $\alpha + tt$ is the sum of n random variables Y_i, $i = 1, \ldots, n$, where Y_i is the value of the flow through the i^{th} path. Y_i is, then, w_i with probability $p_i = \frac{1}{2} - \frac{1}{2\alpha}$, which is equal to the probability that the label l_{e_i} is smaller than the label $l_{e'_i}$, so that the path (e_i, e'_i) is a journey, and is zero otherwise. Then, $v = Y_1 + \ldots + Y_n$ and it holds that $Pr[v \leq C] = Pr[\sum_{i=1}^{n} Y_i \leq C]$.

Now, let J_k be the set of all vectors, $(\rho_1, \ldots, \rho_n)$, of n entries/weights in total, such that each ρ_i is either 0 or the corresponding w_i, and there are exactly k positive entries in the vector. Let $\vec{g} = (g_1, \ldots, g_n)$ be a specific assignment of weights to $Y_1, \ldots, Y_n$, respectively, i.e., $g_i = w_i$ with probability $\frac{1}{2} - \frac{1}{2\alpha}$ and, otherwise, $g_i = 0$; notice that $\vec{g} \in J_k$, for some $k \in \{0, \ldots, n\}$. Then,

$$\begin{aligned} Pr[v \leq C] &= Pr[\sum_{i=1}^{n} Y_i \leq C] \\ &= \sum_{\vec{g}} Pr[Y_i = g_i,\ \forall i = 1, \ldots, n] \cdot x(\vec{g}), \qquad (3.2) \end{aligned}$$

where:

$$x(\vec{g}) = \begin{cases} 1 & \text{, if } \sum_{i=1}^{n} g_i \leq C \\ 0 & \text{, otherwise.} \end{cases}$$

For each particular $\vec{g}$ with exactly k positive weights, the probability that it occurs is $\left(\frac{1}{2} - \frac{1}{2\alpha}\right)^k \left(\frac{1}{2} + \frac{1}{2\alpha}\right)^{n-k}$. So, from Equation 3.2 we get:

$$\begin{aligned} Pr[v \leq C] &= \sum_{k=0}^{n} \sum_{\vec{g} \in J_k} x(\vec{g}) \left(\frac{1}{2} - \frac{1}{2\alpha}\right)^k \left(\frac{1}{2} + \frac{1}{2\alpha}\right)^{n-k} \\ &= \left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n \sum_{k=0}^{n} \sum_{\vec{g} \in J_k} x(\vec{g}) \left(\frac{\frac{1}{2} - \frac{1}{2\alpha}}{\frac{1}{2} + \frac{1}{2\alpha}}\right)^k \end{aligned} \tag{3.3}$$

The following holds (using Bernoulli's inequality):

$$1 \geq \left(\frac{\frac{1}{2} - \frac{1}{2\alpha}}{\frac{1}{2} + \frac{1}{2\alpha}}\right)^k \geq \left(\frac{\frac{1}{2} - \frac{1}{2\alpha}}{\frac{1}{2} + \frac{1}{2\alpha}}\right)^n = \left(\frac{\alpha - 1}{\alpha + 1}\right)^n = \left(1 - \frac{2}{\alpha + 1}\right)^n \geq 1 - \frac{2n}{\alpha + 1} \tag{3.4}$$

Let $T = \sum_{k=0}^{n} \sum_{\vec{g} \in J_k} x(\vec{g})$ and note that T is exactly the number of subsets of $J = \{w_1, \ldots, w_n\}$ with total weight at most C. Then, we get from Equation 3.3 and Relation 3.4:

$$\begin{array}{rcccl} \left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n \left(1 - \frac{2n}{\alpha+1}\right) T & \leq & Pr[v \leq C] & \leq & \left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n T \quad \Leftrightarrow \\ \left(1 - \frac{2n}{\alpha+1}\right) T & \leq & Pr[v \leq C] \frac{1}{\left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n} & \leq & T \quad \Leftrightarrow \\ T - \frac{2nT}{\alpha+1} & \leq & \frac{Pr[v \leq C]}{\left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n} & \leq & T \end{array}$$

Now, assume that $\alpha + 1 > 2nT$; we can guarantee that by selecting α to be, for example, 2^n, or larger. Then, $0 < \frac{2nT}{\alpha+1} < 1$. Let $\varepsilon = \frac{2nT}{\alpha+1}$. Then, we get:

$$T - \varepsilon \leq \frac{Pr[v \leq C]}{\left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n} \leq T$$

Note that $\left(\frac{1}{2} + \frac{1}{2\alpha}\right)^n$ can be represented by a polynomial in n number of bits and can be computed in polynomial time.

If we had a polynomial-time algorithm, A, to exactly compute $Pr[v \leq C]$ for any C and α, then we could exactly compute (also in polynomial time) a number between $T - \varepsilon$ and T, for $0 < \varepsilon < 1$. But, this determines T exactly. So, such an algorithm A would solve a #**P**-complete problem in polynomial time. ■

Remark. If each of the random variables Y_i was of the form $Y_i = w_i$ with probability $p_i = \frac{1}{2}$, and zero otherwise, then the reduction to the KNAPSACK problem would be immediate [49, 71]. However, the possibility of ties in the various l_{e_i} and $l_{e'_i}$s excludes the respective journeys and the reduction does not carry out immediately.

Now, given a mixed temporal network $G(E_1, E_2, \alpha)$, let v be the random variable representing the maximum temporal flow in G.

Definition 3.19. The truncated by B expected maximum temporal flow of $G(E_1, E_2, \alpha)$, denoted by $E[v, B]$, is defined as:

$$E[v, B] = \sum_{i=1}^{B} iPr[v = i]$$

Clearly, it is $E[v] = E[v, +\infty]$.

We are now ready to prove the main theorem of this section:

Theorem 3.20. *It is #**P**-hard to compute the expected maximum truncated Temporal Flow in a Mixed Temporal Network $G(E_1, E_2, \alpha)$.*

Proof. Consider the single-labelled mixed temporal network $G(E_1, E_2, \alpha)$ of Figure 3.8, in which s has n outgoing disjoint directed paths of two edges e_i, e'_i to a node t_1, and then there is an edge from t_1 to t. The capacity of each edge (s, v_i) $, i = 1, \ldots, n$, is w_i, the capacity of each edge (v_i, t_1) $, i = 1, \ldots, n$, is $w'_i \geq w_i$, and the capacity of the edge (t_1, t) is an integer B such that $\frac{1}{2}\sum_{i=1}^{n} w_i < B < \sum_{i=1}^{n} w_i$. The *single label* of edge (t_1, t) is some $b \in \mathbb{N}$, $b > \alpha$, where α is the maximum possible label that the other edges may select; in particular, each of the edges $(s, v_i), (v_i, t_1)$ $, i = 1, \ldots, n$ receives a unique random label drawn uniformly and independently from $\{1, \ldots, \alpha\}$.

Clearly, the maximum temporal flow from s to t until time b is $v' = B$, if $v = \sum_{i=1}^{n} Y_i > B$, and is $v' = v = \sum_{i=1}^{n} Y_i$, otherwise; here Y_i, $i = 1, \ldots, n$, is the random variable representing the flow passing from t to t_1 via v_i in the time until α.

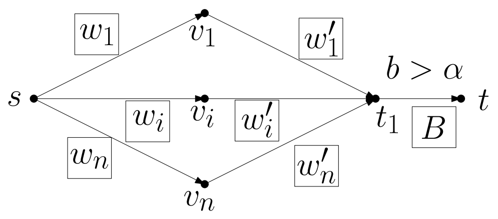


FIGURE 3.8: The mixed temporal network $G(E_1, E_2, \alpha)$, where $E_1 = \{(t_1, t)\}$ with $l_{(t_1,t)} = b > \alpha$.

So, if $E[v']$ is the expected value of v', we have:

$$\begin{aligned} E[v'] &= \sum_{i=0}^{B} iPr[v = i] + B \cdot Pr[v > B] \\ &= E[v, B] + B\,(1 - Pr[v \leq B]) \qquad (3.5) \end{aligned}$$

So, if we had a polynomial-time algorithm that could compute truncated expected maximum temporal flow values in mixed temporal networks, then we could compute $E[v']$ and $E[v, B]$; we could then solve Equation 3.5 for $Pr[v \leq B]$ and, thus, compute it in polynomial time. But to compute $Pr[v \leq B]$ is #**P**-hard by Lemma 3.18. ■

3.5 Open Problems

With regards to the study of temporal flows when the labels are not random, there is still scope to study the case of bounded buffers. It would be an interesting result to prove that the max-flow min-cut theorem, which is one of the most prominent duality theorems known for static graphs, holds in any case (we have shown here that it holds for the case of infinite buffers). A generic method that may be used for obtaining such duality relations is to appropriately express the various temporal problems using an ILP (we have given here a linear program for the problem of finding the maximum temporal flow) and then to try to interpret its dual ILP in terms of temporal graph properties.

There is also scope to make the study of temporal flows more complete, by investigating flows in *periodic* temporal networks; these are networks in which every edge e appears once every x_e time-steps, for some $x_e \in \mathbb{Z}^+$. This is a case that has not been extensively studied in the temporal networks literature, and it would require a careful analysis to investigate problems like the maximum temporal flow that can reach the sink by some particular day. One could perhaps use and generalise known techniques that work for static graphs, or try to develop new algorithmic tools tailored to the specific temporal problem. The application of dynamic programming or divide-and-conquer approaches would be a possible direction, either on the temporal graph itself or on its time-extended static graph; however, in the case of periodic temporal networks, the time-extended graph becomes infinite, so one would have to take that into account and possibly try to extract a different type of time-extended graph, which would be repeatedly used until the day in question.

With regards to the study of mixed and random temporal networks, it is still an open problem if there an FPTAS for the expected maximum flow value. Apart from seeking for efficient approximations for temporal problems that turn out to be hard to solve, an alternative approach that could be used to understand the computational landscape is to restrict the topology of the underlying graph and/or turn the attention towards randomized algorithms.

Chapter 4

Temporal Graph Design for Temporal Connectivity

This chapter covers the full version and the extension of our work published in [6] at the 13th Workshop on Approximation and Online Algorithms, 2015 (WAOA'15). We study here the design of small cost temporally connected temporal graphs, under various constraints. We say that a graph is temporally connected if there is a $u \to v$ journey for any pair of vertices $u, v,\ u \neq v$. We consider both the case where a designer of temporal graphs is free to choose any edge availabilities, given an underlying graph, and the case where one is given a temporal graph (already labelled) and is only allowed to use the given labelling or a subset of it. In both those cases, our goal is to achieve designs of small cost that also are temporally connected. We define the cost of a labelling (also cost of the corresponding temporal graph) to be the *total* number of labels assigned to the edges of the underlying graph. It is worth noting that both of the cases we consider can be motivated by realistic scenarios. This is clearer in the occasion where one is free to design the temporal graph from the very beginning, where applications can be found in transport network design. However, the case where one can only use an already labelled temporal graph can also find motivation in practical scenarios; as an example, think of phone companies who "borrow" network space from other phone companies and, thus, must use whatever availabilities the pre-existing company has to offer.

4.1 Overview

4.1.1 Motivation

As in the previous Chapters, this Chapter focuses on *discrete time* following the model of [69, 81]. In several dynamic settings from which temporal graphs may be motivated, maintaining connections might come at a cost; consider a transport network or an unstable chemical or physical structure, where money or energy is required to keep a connection available. We define the cost as the total number of discrete time instances at which the network edges become available. We also define temporal connectivity in a temporal graph as the existence of journeys between all pairs of vertices in the graph.

If one has absolute freedom to design a small cost temporally connected temporal graph on an underlying static graph, i.e., choose the edge availabilities, then a reasonable design would be to select a rooted spanning tree and choose appropriate availabilities to construct time-respecting paths from the leaves to the root and *then* from the root back to the leaves. However, in more complicated scenarios one may not be free to *choose* edge availabilities arbitrarily but instead *specific* link availabilities may pre-exist for the network; then, one is able to design a temporally connected temporal network using only the pre-existing availabilities or a subset of them. Imagine a hostile network on a complete graph where availability of a link means a break in its security, e.g., when the guards change shifts, and only then are we able to pass a message through the link. So, if we wish to send information through the network, we may only use the times when the shifts change and it is reasonable to try and do so by using as few of these breaks as possible. In such scenarios, we may need to first verify that the pre-existing edge availabilities indeed define a temporally connected temporal graph. Then, we may try to reduce the cost of the design by *removing* unnecessary (redundant) edge availabilities if possible, without losing temporal connectivity. Consider, again, the clique network of n vertices with one time availability per edge; it is clearly temporally connected with cost $\Theta(n^2)$. However, it is not straightforward if all these edge availabilities are necessary for temporal connectivity. We resolve here the complexity of finding the maximum number of redundant labels in any given temporal graph.

4.1.2 Definitions

As previously mentioned, the availability of links in many networks and several applications comes at a cost. For example, in secure networks there is a cost (per discrete time instance) to keep a link secure. We abstract such considerations by the concept of the *cost* of a temporal graph and wish to have temporal graphs of low cost.

Definition 4.1 (Cost of a labelling)**.** Let $G(L) = (V, E, L)$ be a temporal (di)graph and L be its labelling. The *cost* of L is defined as $c(L) = \sum_{e \in E} |L_e|$.

We also focus here on temporal graphs that are *temporally connected*, i.e., have the following property:

Definition 4.2 (Property TC)**.** A temporal (di)graph $G(L) = (V, E, L)$ satisfies the property TC, or equivalently L satisfies TC on G, if for any pair of vertices $u, v \in V,\ u \neq v$, there is a $u \to v$ journey *and* a $v \to u$ journey in $G(L)$. A temporal (di)graph that satisfies the property TC is called *temporally connected.*

Example. An undirected complete graph, K_n, is temporally connected under any labelling L with $L_e \neq \emptyset$ for every $e \in E(K_n)$. Indeed, there is a $u \to v$ journey and a $v \to u$ journey between any $u, v \in V(K_n),\ u \neq v$, namely the time edge (u, v, l) and the time edge (v, u, l) respectively, for any $l \in L_{\{u,v\}}$.

Definition 4.3 (Minimal temporal graph)**.** A temporal graph $G(L) = (V, E, L)$ over a (strongly) connected (di)graph is *minimal* if $G(L)$ has the property TC, and the removal of any label from any $L_e,\ e \in E$, results in a $G(L')$ that *does not* have the property TC.

Definition 4.4 (Removal profit)**.** Let $G(L) = (V, E, L)$ be a temporally connected temporal graph. The *removal profit* $r(G, L)$ is the largest total number of labels that can be removed from L without violating TC on G.

Here, removal of a label l from L refers to the removal of l only from a particular edge and not from all edges that are assigned label l, i.e., if $l \in L_{e_1} \cap L_{e_2}$ and we remove l from both L_{e_1} and L_{e_2}, it counts as two labels removed from L.

Notice that if many edges have the same label, we can encounter *trivial cases* of minimal temporal graphs. For example, the complete graph where every edge appears at time, say $t = 5$, is minimal but there are no journeys of length larger than 1. To avoid cases where minimality is caused merely due to the assignment of the same label(s) to many (or all) edges, we will often consider a special subcategory of (single-labelled) temporal graphs:

Definition 4.5 (SLSE temporal graphs)**.** A Single-label-single-edge (SLSE) temporal graph is a temporal graph, each edge of which has a single label and no two edges have the same label, i.e., each label is assigned to (at most) a single edge. A labelling that gives an SLSE temporal graph is also called *SLSE labelling.*

4.1.3 Previous work and our contribution

The model we consider, as mentioned in previous Chapters, is very closely related to the single-labelled model of the seminal paper of Kempe et al. [69] as well as the multi-labelled model of Mertzios et al. [81]. Kempe et al. [69] consider the case of one *real* label per edge and examine how basic graph properties change when we impose the temporal condition; here, we extend that model by considering multiple labels per edge but we restrict our focus to integer labels. Mertzios et al. [81] also extended the model of [69] to many labels per edge and mainly examined the number of labels needed for a temporal design of a network to guarantee several graph properties with certainty. They also defined the cost notion and, amongst other results, gave an algorithm to compute foremost journeys which can be used to decide property TC. However, the time complexity of that algorithm was pseudo-polynomial, as it was dominated by the cube of the maximum label used in the given labelling.

In fact, the problem of testing whether a dynamic graph is temporally connected has been studied before in various settings [16, 25, 100]. Bui-Xuan et al. [25] propose an algorithm for computing foremost journeys in a model of evolving graphs, where nodes and edges are associated with lists of time intervals, representing their existence over time, and each edge has a traversal time. In a similar setting, Whitbeck et al. [100] study temporal reachability graphs, in which a (u, v)-edge is present at time t if (in the corresponding time-varying graph) there is a $u \to v$ journey leaving u after t and arriving at v after at most some specified time-interval. Barjon et al. [16] investigate discrete-time evolving graphs, for which they compute the *transitive closure of journeys*, i.e., a static directed graph whose edges represent potential journeys. The algorithm they propose depends on the maximum label used, the number of vertices, and the maximum number of edges that simultaneously exist.

Here, we show that if the designer of a temporal graph can select edge availabilities freely, then an asymptotically optimal linear-cost (in the size of the graph) design that satisfies TC can be easily obtained (see Section 4.3). We give a matching lower bound to indicate optimality, in the case where the underlying graph is a tree. However, there are pragmatic cases where one is not free to design a temporal graph anew; instead, one is *given* a set of possible availabilities per edge with the claim that they satisfy TC and the constraint that one may only use them or a subset of them for their design. We also propose a simple algorithm to verify TC in low polynomial time (see Section 4.2). The *given* design may also be minimal; we partially characterise minimal designs in Section 4.4. On the other hand, there may be some labels of the initial design that can be removed without violating TC (and also result in a lower cost). In this case, how many labels can we remove at best? Our main technical result is that this problem is APX-hard, i.e. it has no PTAS unless $P = NP$. On the positive side, we show that in the case of complete graphs and random graphs, if the labels are also assigned at random, there is asymptotically almost surely a very large number of labels that can be removed without violating TC. A preliminary version of this work appeared in the 13th Workshop on Approximation and Online Algorithms, WAOA 2015 [6].

Stochastic aspects and/or survivability of network design were also considered in the work of Gupta et al. [57], the work of Lau and Singh [78], and the work of Lau et al. [77].

4.2 A low polynomial time algorithm for deciding TC

In this section, we propose a simple polynomial-time algorithm which, given a temporal (di)graph $G(L) = (V, E, L)$ and a source vertex $s \in V$, computes a *foremost* $s \to v$ journey, for every $v \neq s$, if such a journey exists.

Theorem 4.6. *Algorithm 2 correctly computes for every vertex $v \in V$, $v \neq s$ a foremost journey from s to v, if one exists. The time complexity of the algorithm is $O(c(L) \cdot \log c(L))$.*

Proof. The algorithm actually considers each existing label in the sequence of time labels, from the smallest to the largest one. It examines each time edge (resp. time arc) exactly once. For each label considered, it computes the foremost journeys from s which arrive at that time.

Consider the point where every time-edge with label smaller than some label t in the order given by S' has been examined and let t be the next label in the order to be examined by the algorithm. Assume that at that point, all vertices in R have been reached by s via foremost journeys. Let (u, v, t) be a time-edge such that $u \in R$, with $arrival\_time[u] < t$, and $v \notin R$ and let $f(s, u)$ be the foremost journey from s to u. Clearly, $J = (f(s, u), (u, v, t))$ is a journey from s to v; denote the respective arrival time by $a(J)$. We claim that J is a foremost journey from s to v. To see that this claim holds assume that there is some other journey J' from s to v such that $a(J') < a(J)$. So there must be some time-edge (w, z, t') in J' for some $w \in R$, $z \notin R$, and $t' < t$. But the latter contradicts the fact that z must be in R, since the algorithm should have added it earlier when examining time-edges with label t'. The proof follows by induction on $t = l_{min}$ (the smallest label in the temporal graph) at which time $R = \{s\}$ (s has trivially been reached by a foremost journey from itself, so the claim holds for the base case).

If a journey from s to v does not exist, then in any path from s to v in G there is a pair of consecutive edges with equal or decreasing labels; let the first such pair be edges $\{x, y\}$ with label l, and $\{y, r\}$ with label $l' \leq l$. If $l = l'$ the algorithm will add y to R but it will skip all subsequent edges in the path, since the *if* statement in line 10 will read *false*. Therefore, vertices that cannot be reached by s will have infinite arrival time after the termination of the algorithm.

The time complexity of the algorithm is dominated by the sorting time of $S(L)$. One can sort $S(L)$ by comparison-based sorting resulting in running time $O(|S(L)| \cdot \log |S(L)|) = O(c(L) \cdot \log c(L))$. ■

Conjecture. We conjecture that any algorithm that computes journeys out of a vertex s must sort the time edges (resp. time arcs) by their labels, i.e., we conjecture that Algorithm 2 is asymptotically optimal with respect to the running time.

Algorithm 2: Foremost journeys algorithm

```
Input: A temporal (di)graph $G(L) = (V, E, L)$ of $n$ vertices, the set of all time
       edges (arcs) of which is denoted by $S(L)$; a designated source vertex
       $s \in V$
Output: A foremost $s \to v$ journey from $s$ to all $v \in V \setminus \{s\}$, where such a
        journey exists; if no $s \to v$ journey exists, then the algorithm reports it.
1  Sort $S(L)$ in increasing order of labels ;        /* Note that $|S(L)| = c(L)$ */
2  Let $S'$ be the sorted array of time edges (resp. time arcs) according to time
   labels;
3  $R := \{s\}$ ;   /* The set of vertices to which $s$ has a foremost journey
   */
4  $arrival\_time[s] := 0$;
5  for all $v \in V \setminus \{s\}$ do
6      $parent[v] := \emptyset$;
7      $arrival\_time[v] := +\infty$;
8  end for
9  for all time edges (resp. time arcs) $(a, b, l)$ in the order given by $S'$ do
10     if $a \in R$ and $b \notin R$ and $arrival\_time[a] < l$ then
11         $parent[b] := a$;
12         $arrival\_time[b] := l$;
13         $R := R \cup \{b\}$;
14     end if
15 end for
```

Note that Algorithm 2 can even compute foremost $s \to v$ journeys, if they exist, that *start* from a given time $t_{start} > 0$. Simply, one ignores the time edges (arcs) with labels smaller than the start time.

4.3 Asymptotically cost-optimal design for TC in undirected graphs.

In this section, we study temporal design issues on connected undirected graphs, so that the resulting temporal graphs are temporally connected. In this scenario, the designer has absolute freedom to choose the edge availabilities of the underlying graph.

Lemma 4.7. *There is an infinite family of graphs G_n of n vertices, for which the cost of any labelling that satisfies TC is at least $2n - 3$.*

Proof. Consider the star graph of n vertices, $n \geq 4$. Let v_n be the root and $v_1, v_2, \ldots, v_{n-1}$ be the leaves. In any labelling on the star graph, which assigns only one label to two (or more) edges $(v_n, v_x),\ (v_n, v_y),\ x, y = 1, 2, \ldots, n-1,\ x \neq y$, at least one of the vertices v_x, v_y cannot reach the other via a journey. Therefore, any TC satisfying labelling on the star graph must assign at least 2 labels to all edges of the graph, except possibly on one edge where it assigns a single label. The TC satisfying labelling which assigns labels $1, 3$ to all edges except for one and label 2 to the remaining edge has, therefore, minimum cost, namely $2n - 3$ (see Figure 4.1). ∎

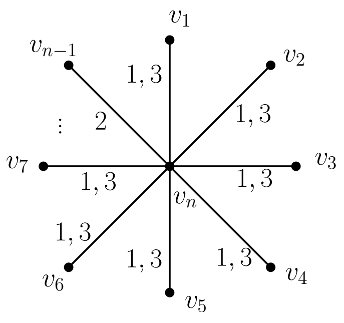


FIGURE 4.1: Labelling a star graph in an optimal way

In fact, the result of Lemma 4.7 is optimal for any tree. Theorem 4.8 shows a lower bound for trees and an asymptotically optimal[1] way of labelling any connected undirected graph to satisfy TC.

Theorem 4.8. *(a) For any tree $G = (V, E)$ of n vertices and for any labelling L that satisfies the property TC on G, the cost of L is $c(L) \geq 2n - 3$.*

(b) Given a connected undirected graph $G = (V, E)$ of n vertices, we can design a labelling L of cost $c(L) = 2n - 3$ that satisfies the property TC on G. L can be computed in polynomial time.

Proof. (a) Any TC satisfying labelling on G must assign at least 1 label to each edge of G; otherwise, one endpoint of an edge without labels would never reach the other.

Now, assume that there is a TC satisfying labelling L which assigns to two (or more) edges $\{u, x\}, \{y, v\} \in E(G)$ exactly one label; assume without loss of generality that u and v are the endpoints with the largest pairwise

[1]Any connected undirected graph needs at least $n - 1$ labels on its edges to be temporally connected, and we show a TC satisfying labelling of $2n - 2 = \Theta(n)$ labels.

distance between the vertices of those two edges. Then, if $l_{\{u,x\}} < l_{\{y,v\}}$, v cannot reach u in $G(L)$, and if $l_{\{u,x\}} \geq l_{\{y,v\}}$, u cannot reach v in $G(L)$ (that is due to the uniqueness of the path between the vertices u and v in G).

Therefore, any labelling that satisfies TC on G must assign at least two labels to all edges of G, except maybe one edge which may have a single label.

(b) Consider a fixed, but arbitrary, spanning tree T of G and let a node, w, of degree 1 be the root of T. Also let w' be the single child of w in T and denote by T' the subtree of T that is rooted at w'. Let r be the length of the longest path from w' to any leaf of T', i.e., r is the radius of T'. We assign labels to the edges of T as follows:

Going upwards. Any edge of T' incident to a leaf gets label 1. Any edge $e = \{u, v\}$ of T', with $d(w', v) = d(w', u) + 1$, where the subtree T^* rooted at v has been labelled going upwards towards w', gets a label $l_e = max\{\text{all labels in } T^*\} + 1$ (cf. Figure 4.2).

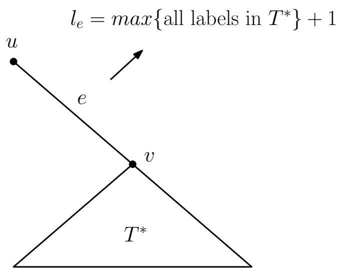


FIGURE 4.2: Labelling "going upwards" to the root

The edge $\{w, w'\}$ We label the edge $\{w, w'\}$ of T with the single label $r+1$.

Going downwards. Any edge of T' incident to w' gets a label $r + 2$. Any edge e of T' in a path from w' to a leaf of T', the *parent edge*[2] of which has been labelled, going downwards, with label l', gets a label $l_e = l'+1$.

We can easily implement the above process by topologically ordering the vertices of T in levels using *Breadth First Search* and implement the "going upwards" and "going downwards" procedures accordingly. The above method results in a labelling where:

1. each edge of T has 2 labels, except for the edge $\{w, w'\}$, which has a single label,

[2]The edge before it in the sequence of edges from the root w' to the respective leaf.

2. each edge of $E \setminus T$ has no label and
3. for each ordered pair of vertices $u, v \in V,\ u \neq v$, there is a (u, v)-journey.

To show 3, just notice that one can go from any vertex $u \in V$ to any other vertex $v \in V$ by going up in T from u to w and then going down in T from w to v via strictly increasing labels, by construction.

■

Example. Figure 4.3 shows an example of the procedure described above. Notice the existence of journeys from any vertex to every other vertex in the resulting temporal graph.

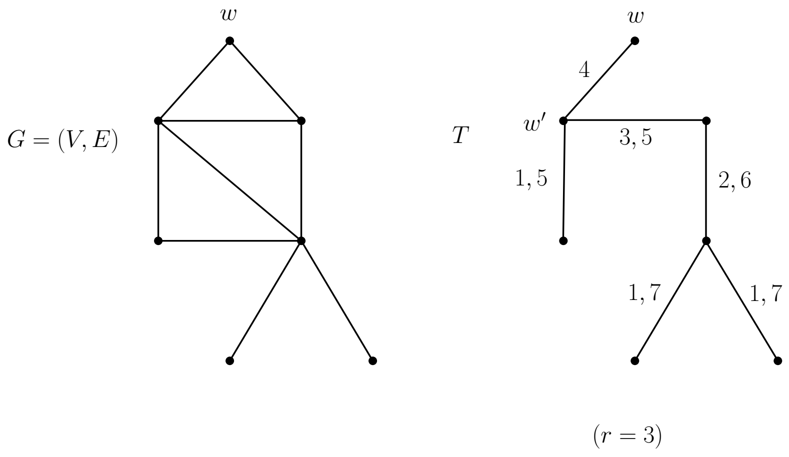


FIGURE 4.3: Labelling a connected undirected graph to satisfy TC

Conjecture. We conjecture that for any connected undirected graph G of n vertices and for any labelling L that satisfies the property TC on G, the cost of L is $c(L) \geq 2n - 4$.

4.4 Minimal Temporal Designs

Suppose now that a temporal graph on a (strongly) connected (di)graph $G = (V, E)$ is *given* to a designer with the claim that it satisfies TC. In this scenario, the designer is allowed to only use the given set of edge availabilities, or a subset of them. If the given design is not minimal, they may wish to remove as many labels as possible, thus reducing the cost. Minimality of a design can be verified by running Algorithm 2 (see Section 4.2) for every $s \in V$.

4.4.1 A partial characterisation of minimal temporal graphs

As mentioned earlier, if many edges have the same label, we can encounter *trivial cases* of minimal temporal graphs. To avoid such cases, we focus our attention here to the class of SLSE temporal graphs, in which every edge only becomes available at one moment in time and no two different edges become available at the same time. Are there minimal SLSE temporal graphs with non linear (in the size of the graph) cost? For example, any complete SLSE temporal graph satisfies TC. Are all these $\Theta(n^2)$ labels needed for TC, i.e., are there minimal temporal complete graphs? As we prove in Theorem 4.13, the answer is negative. However, we give below a minimal temporal graph on n vertices with non-linear in n cost, namely with $O(n \log n)$ labels.

4.4.1.1 A minimal temporal design of $n \log n$ cost

Definition 4.9 (Hypercube graph)**.** The k-hypercube graph, commonly denoted Q_k, is a k-regular graph of 2^k vertices and $2^{k-1} \cdot k$ edges. The 1-hypercube is the graph of two vertices and one edge. Recursively, the n-hypercube is produced by taking two isomorphic copies of the $(n-1)$-hypercube and adding edges between the corresponding vertices.

Definition 4.10 (Flat)**.** In geometry, a *flat* is a subset of the n-dimensional space that is congruent to a Euclidean space of lower dimension, e.g., the flats in the two-dimensional space are points and lines. In the n-dimensional space, there are flats of every dimension from 0, i.e., points, to $n-1$, i.e., hyperplanes.

Theorem 4.11. *There exists an infinite class of minimal temporal graphs on n vertices with $\Theta(n \cdot \log n)$ edges and $\Theta(n \cdot \log n)$ labels, such that different edges have different labels.*

Proof. We present a minimal temporal graph on the hypercube graph of n vertices. Consider Protocol 3 for labelling the edges of $G = Q_k = (V, E)$. The temporal graph, $G(L)$, that this labelling procedure produces on the hypercube is minimal. Indeed, first we will prove that the temporal graph produced by Protocol 3 satisfies TC on $G = Q_k$.

Consider vertices $u, v \in V$ and the steps described in Protocol 4 to reach v, starting from u, via temporal edges. The procedure described in Protocol 4 gives a journey from u to v, which is also *unique*. It suffices to consider the k-bit binary representation of the vertices of G. Notice that if the hamming distance of the labels of two vertices $u, v \in V(G)$ is exactly m, then to reach v from u via a temporal path in the temporal graph on G, we need to move through vertices by consecutively swapping the bits in which u and v differ in the order of dimensions. This way, we maintain the strictly increasing order of the time labels we use and, swap by swap, we approach the destination. Note also that swapping only the bits in which u and v differ is the only way to not violate the increasing order of time labels we use: without loss of generality, suppose that the j^{th} bit of u is 1 and so is j^{th} bit of v. If, starting from u, we swap the j^{th} bit to 0, i.e., we use an edge, e, on the j^{th} dimension, then at a future step, we again need to swap the j^{th} bit back to 1 (otherwise, we never reach v). However, the two swaps cannot be consecutive, because then we would use edge e twice and we violate the increasing order of labels. So, we would need to move to a higher dimension after the first of the two swaps; but, then, we have used labels that are larger than all the labels of the j^{th} dimension, so using any edge of the j^{th} dimension would also violate the increasing order of labels.

Protocol 3: Labelling the hypercube graph, $G = Q_k$

```
1  Consider the $k$ dimensions of the hypercube $G = Q_k$, $x_1, x_2, \dots, x_k$;
2  for $i = 1 \dots k$ do
3      Let $X_i := \{e_{i1}, e_{i2}, \dots, e_{i2^{k-1}}\}$ be the list of edges in dimension $x_i$, in an
       arbitrary order;
4      Let $L_i$ be the (sorted from smallest to largest) list of labels
       $L_i := \{(i-1) \cdot 2^{k-1} + 1, (i-1) \cdot 2^{k-1} + 2, \dots, i \cdot 2^{k-1}\}$ ;
5  end for
6  for $i = 1 \dots k$ do
7      for $j = 1 \dots 2^{k-1}$ do
8          Assign the (current) first label of $L_i$ to the (current) first edge of $X_i$ ;
9          Remove the (current) first label of $L_i$ from the list;
10         Remove the (current) first edge of $X_i$ from the list;
11     end for
12 end for
13 return the produced temporal graph, $G(L)$;
```

Since our labelling gives a *unique* $u \to v$ journey, for every $u, v \in V$, and since all labels assigned to the edges of E are used in the union of all those journeys, the deletion of any single label will violate TC. Therefore, $G(L)$ is minimal. Finally,

```
Protocol 4: A temporal path from $u$ to $v$ in the temporal graph on $G = Q_k$
Input: The considered temporal graph on the hypercube $G = Q_k$, vertices
       $u, v \in V(G)$
Output: Array $x$ of vertices, which the $u \to v$ journey passes through
1 $x[0] := u$;
2 Find the flat of the smallest dimension, $m$, which both $u$ and $v$ lie on;
3 Consider the increasing order of the $m$ dimensions in that flat: $d[1], d[2], \dots, d[m]$;
4 for $i = 1 \dots m$ do
5     Use the incident edge of $x[i-1]$ that lies on dimension $d[i]$ and let $x[i]$ be the
      other endpoint of that edge;
6 end for
```

note that the temporal graph $G(L)$ on the hypercube graph $G = Q_k$ has $n = 2^k$ vertices, $\frac{1}{2}n \cdot \log n$ edges and $\frac{1}{2}n \cdot \log n$ labels. ■

4.4.1.2 A minimal temporal design of linear in n cost

In the previous section, we showed that there are graphs of non-linear cost (in the number of vertices) that are minimal. Here, we show that there are classes of minimal graphs whose cost is linear in the number of their vertices.

Indeed, as seen in Lemma 4.7 (Section 4.3), the star graph of n vertices needs at least $\Theta(n)$ labels to satisfy TC and, in fact, we present there a TC satisfying labelling of $\Theta(n)$ labels (see Figure 4.1). Theorem 4.8(b) (Section 4.3) also gives a class of minimal temporal graphs of linear cost in the number of vertices. Therefore, we have the following Corollary:

Corollary 4.12. *There exists an infinite class of minimal temporal graphs on n vertices with $\Theta(n)$ edges and $\Theta(n)$ labels.*

4.4.1.3 SLSE Cliques of at least 4 vertices are not minimal

The complete graph on n vertices, K_n, with an SLSE labelling L, i.e., a labelling that assigns a single label per edge, different labels to different edges, is an interesting case, since $K_n(L)$ always satisfies TC. However, it is not minimal as the theorem below shows.

Theorem 4.13. *Let $n \in \mathbb{N}$, $n \geq 4$ and denote by K_n the complete graph on n vertices. There exists* no minimal *SLSE temporal graph on $K_n(L)$. In fact, we can* *remove (at least)* $\lfloor \frac{n}{4} \rfloor$ *labels from any SLSE labelling on* $K_n(L)$ *without violating TC.*

Proof. The proof is divided in two parts, as follows:

(a) We first show that any SLSE labelling on the complete graph on 4 vertices produces a temporal graph that is not minimal, i.e., the theorem holds for K_4. Consider the six different labels a, b, c, d, x, y assigned by an SLSE labelling to the edges of K_4 as shown in Figure 4.4.

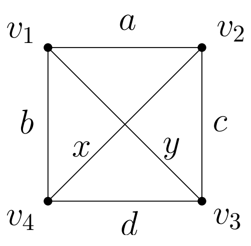


FIGURE 4.4: Any SLSE labelling on K_4 is not minimal.

Up to their renaming, there are three possible cases for the labels a, b, c, d. Counting all the cases of *alternation*, *cycle*, and *entanglement* (see below) would give us all possible $4! = 24$ cases.

1. (Alternation) $a < b > d < c > a$.
   It is easy to see that in this case, both diagonals can be removed: v_1 can reach v_3 using labels a and then c; v_3 can reach v_1 using labels d and then b; v_2 can reach v_4 using labels a and then b; v_4 can reach v_2 using labels d and then c.
2. (Cycle) $a < b < d < c$.
   Here, diagonal x can be removed: v_2 can reach v_4 using labels a and then b; v_4 can reach v_2 using labels d and then c.
3. (Entanglement) $a < b < c < d$.
   This is a more complex case, for which we distinguish the following five sub-cases:
   i) $x < b$ and $y < c$.
   We can remove label a: v_1 can reach v_2 using labels y and then c; v_2 can reach v_1 using labels x and then b.
   ii) $x < b$ and $y > c$.
   We can remove label b: v_1 can reach v_4 using labels a, then c and

then d; v_4 can reach v_1 using labels x, then c and then y (notice that $x < b < c < y$).

iii) $x > b$ and $y > c$.
We can remove label a: v_1 can reach v_2 using labels b and then x; v_2 can reach v_1 using labels c and then y.

iv) $x > b$ and $b < y < c$.
We can remove label x: v_2 can reach v_4 using labels a and then b; v_4 can reach v_2 using labels b, then y and then c.

v) $x > b$ and $y < b$.
We can remove label c: v_2 can reach v_3 using labels a, then b and then d; v_3 can reach v_2 using labels y, then b and then x.

Notice that the coverage of the above five cases is complete (see Figure 4.5).

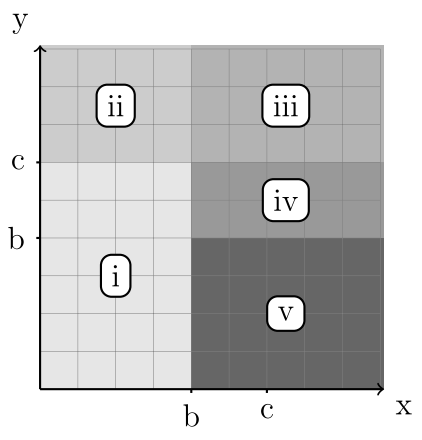


FIGURE 4.5: The six sub-cases cover all possible scenarios of "entanglement".

(b) Now, consider the complete graph on $n \geq 4$ vertices, $K_n = (V, E)$. Partition V arbitrarily into $\lceil \frac{n}{4} \rceil$ subsets $V_1, V_2, \ldots, V_{\lceil \frac{n}{4} \rceil}$, such that $|V_i| = 4, \forall i = 1, 2, \ldots, \lceil \frac{n}{4} \rceil - 1$ and $|V_{\lceil \frac{n}{4} \rceil}| \leq 4$. In each 4-clique defined by V_i, $i = 1, 2, \ldots, \lfloor \frac{n}{4} \rfloor$, we can remove a "redundant" label, as shown in (a). The resulting temporal graph on K_n still preserves TC since for every ordered pair of vertices $u, v \in V$:

- if u, v are in the same V_i, $i = 1, 2, \ldots, \lfloor \frac{n}{4} \rfloor$, then there is a $u \to v$ journey that uses time edges within the 4-clique on V_i, as proven in ((a)).
- if $u \in V_i$ and $v \in V_j$, $i \neq j$, then there is a $u \to v$ journey that uses the (direct) time edge on $\{u, v\}$.

■

4.4.2 Computing the removal profit is APX-hard

Note that it is straightforward to check in polynomial time whether a given L satisfies TC on a given (di)graph G, by just checking for every possible (ordered) pair (u, v) of vertices in G whether there is a $u \to v$ journey in $G(L)$. Recall that the removal profit is the largest number of labels that can be removed from a temporally connected graph without destroying TC. We now show that it is hard to approximate the value of the removal profit arbitrarily well for an arbitrary graph, i.e., there exists no PTAS[3] for this problem, unless P=NP. It is worth noting here that, in our hardness proof below, we consider *undirected* graphs; the fact that all $u \to v$ journeys, $u \neq v$ exist in any given (unlabelled) connected undirected graph makes the reduction and the analysis much more involved.

We prove our hardness result by providing an approximation preserving polynomial reduction from a variant of the maximum satisfiability problem, namely from the *monotone Max-XOR(*3*)* problem. Consider a monotone XOR-boolean formula ϕ with variables $x_1, x_2, \ldots, x_n$, i.e., a boolean formula that is the conjunction of XOR-clauses of the form $(x_i \oplus x_j)$, where no variable is negated. The clause $\alpha = (x_i \oplus x_j)$ is XOR-satisfied by a truth assignment τ if and only if $x_i \neq x_j$ in τ. The number of clauses of ϕ that are XOR-satisfied in τ is denoted by $|\tau(\phi)|$. If every variable x_i appears in exactly r XOR-clauses in ϕ, then ϕ is called a *monotone XOR(r)* formula. The *monotone Max-XOR(r)* problem is, given a monotone XOR(r) formula ϕ, to compute a truth assignment τ of the variables $x_1, x_2, \ldots, x_n$ that XOR-satisfies the largest possible number of clauses, i.e., an assignment τ such that $|\tau(\phi)|$ is maximized. The monotone Max-XOR(3) problem essentially encodes the *Max-Cut* problem on 3-regular (i.e., cubic) graphs, which is known to be APX-hard [9].

Lemma 4.14. *[9] The monotone Max-XOR(*3*) problem is APX-hard.*

Now we provide our reduction from the monotone Max-XOR(3) problem to the problem of computing $r(G, L)$. Let ϕ be an arbitrary monotone XOR(3) formula with n variables $x_1, x_2, \ldots, x_n$ and m clauses. Since every variable x_i appears in ϕ in exactly 3 clauses, it follows that $m = \frac{3}{2}n$. We will construct from ϕ a graph $G_\phi = (V_\phi, E_\phi)$ and a labelling L_ϕ of G_ϕ.

[3]PTAS stands for Polynomial-Time Approximation Scheme.

First we construct for every variable x_i, where $1 \leq i \leq n$, the gadget-graph $G_{\phi,i}$ together with a labelling $L_{\phi,i}$ of its edges, as illustrated in Figure 4.6. In this figure, the labels of every edge in $L_{\phi,i}$ are drawn next to the edge. We call the induced subgraph of $G_{\phi,i}$ on the 4 vertices $\{s^{x_i}, u_0^{x_i}, w_0^{x_i}, v_0^{x_i}\}$ the *base* of $G_{\phi,i}$. Moreover, for every $p \in \{1, 2, 3\}$, we call the induced subgraph of $G_{\phi,i}$ on the 4 vertices $\{t_p^{x_i}, u_p^{x_i}, w_p^{x_i}, v_p^{x_i}\}$ the *pth branch* of $G_{\phi,i}$. Finally, we call the edges $\{u_0^{x_i}, w_0^{x_i}\}$ and $\{w_0^{x_i}, v_0^{x_i}\}$ the *transition edges* of the base of $G_{\phi,i}$ and, for every $p \in \{1, 2, 3\}$, we call the edges $\{u_p^{x_i}, w_p^{x_i}\}$ and $\{w_p^{x_i}, v_p^{x_i}\}$ the *transition edges* of the pth branch of $G_{\phi,i}$. For every $p \in \{1, 2, 3\}$ we associate the pth appearance of the variable x_i with the pth branch of $G_{\phi,i}$.

We continue the construction of $G_{\phi,i}$ and $L_{\phi,i}$ as follows. First, we add an edge between any possible pair of vertices $w_p^{x_i}, w_q^{x_j}$, where $p, q \in \{0, 1, 2, 3\}$ and $i, j \in \{1, 2, \ldots, n\}$, and we assign to this new edge $e = \{w_p^{x_i}, w_q^{x_j}\}$ the unique label $L_\phi(e) = \{7\}$. Note here that we add this edge $\{w_p^{x_i}, w_q^{x_j}\}$ also in the case where $i = j$ (and $p \neq q$).

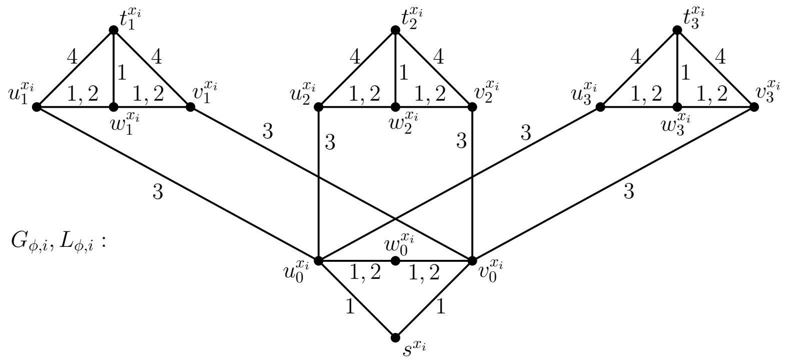


FIGURE 4.6: The gadget $G_{\phi,i}$ for the variable x_i.

Intuitively, the base of $G_{\phi,i}$ (see Figure 4.6) corresponds to the variable x_i and, for every $p \in \{1, 2, 3\}$, the pth branch of $G_{\phi,i}$, together with the two edges $\{u_0^{x_i}, u_p^{x_i}\}$ and $\{v_0^{x_i}, v_p^{x_i}\}$, correspond to the clause of ϕ in which x_i appears for the pth time in ϕ.

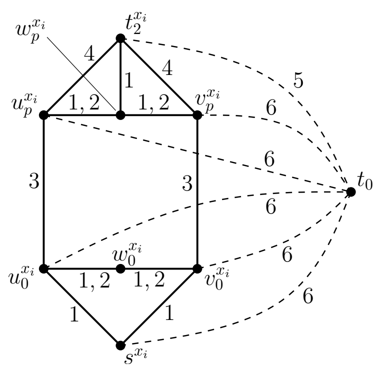
Consider now a clause $\alpha = (x_i \oplus x_j)$ of ϕ. Assume that the variable x_i (resp. x_j) of α corresponds to the pth (resp. to the qth) appearance of x_i (resp. of x_j) in ϕ. Then we identify the vertices $u_p^{x_i}, v_p^{x_i}, w_p^{x_i}, t_p^{x_i}$ of the pth branch of $G_{\phi,i}$ with the vertices $v_q^{x_i}, u_q^{x_i}, w_q^{x_i}, t_q^{x_i}$ of the qth branch of $G_{\phi,j}$, respectively (see Figure 4.7b). Now we add an edge between any possible pair of vertices $t_p^{x_i}, t_q^{x_j}$, $i, j \in \{1, 2, \ldots, n\}$, and $p, q \in \{1, 2, 3\}$. We assign to this new edge $e = \{t_p^{x_i}, t_q^{x_j}\}$ the unique label $L_\phi(e) = \{7\}$.

Furthermore, for every $i \in \{1, 2, \ldots, n\}$ and every $p \in \{1, 2, 3\}$ we define for simplicity of notation the temporal paths $P_{i,p} = (s^{x_i}, u_0^{x_i}, u_p^{x_i}, t_p^{x_i})$ and $Q_{i,p} = (s^{x_i}, v_0^{x_i}, v_p^{x_i}, t_p^{x_i})$.

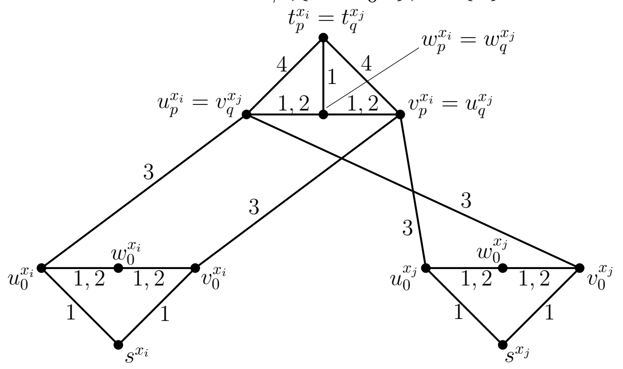
The intuition behind the composition of the gadget-graphs $G_{\phi,i}$ (see Figure 4.7b) is the following. If variable x_i is false in a truth assignment τ of ϕ, then all edges of the paths $P_{i,1}, P_{i,2}, P_{i,3}$ keep their labels as in L_ϕ. Otherwise, if x_i is true in τ, then all edges of the paths $Q_{i,1}, Q_{i,2}, Q_{i,3}$ keep their labels as in L_ϕ. Furthermore, depending on the value of x_i in the assignment τ, each of the transition edges $\{u_p^{x_i}, w_p^{x_i}\}$ and $\{w_p^{x_i}, v_p^{x_i}\}$, where $p \in \{1, 2, 3\}$, keeps exactly one of its two labels from L_ϕ. Consider now a clause $\alpha = (x_i \oplus x_j)$ of ϕ which corresponds to the pth branch of $G_{\phi,i}$ and to the qth branch of $G_{\phi,j}$. Then the only case where *both* edges $\{t_p^{x_i}, u_p^{x_i}\}$ and $\{t_p^{x_i}, v_p^{x_i}\}$ keep their labels from L_ϕ, is when the two variables x_i, x_j have *equal* truth value in the corresponding truth assignment τ of ϕ; that is, when the clause $\alpha = (x_i \oplus x_j)$ is *not* XOR-satisfied by τ. Therefore, intuitively, by a careful counting of the labels it turns out that, if more clauses can be satisfied by a truth assignment τ, then a TC preserving sub-labelling L of L_ϕ can be constructed which avoids more labels from L_ϕ, and vice versa (see Theorem 4.18).

To finalize the construction of the graph G_ϕ, we add a new vertex t_0 to ensure the existence of a temporal path between each pair of vertices of G_ϕ, as follows. This new vertex t_0 is adjacent to vertex $w_0^{x_n}$ and to all vertices in the set $\{s^{x_i}, t_1^{x_i}, t_2^{x_i}, t_3^{x_i}, u_p^{x_i}, v_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$. First we assign to the edge $\{t_0, w_0^{x_n}\}$ the unique label $L_\phi(\{t_0, w_0^{x_n}\}) = \{5\}$. Furthermore, for every vertex $t_p^{x_i}$, where $1 \leq i \leq n$ and $1 \leq p \leq 3$, we assign to the edge $\{t_0, t_p^{x_i}\}$ the unique label $L_\phi(\{t_0, t_p^{x_i}\}) = \{5\}$. Finally, for each of the vertices $z \in \{s^{x_i}, u_p^{x_i}, v_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$ we assign to the edge $\{t_0, z\}$ the unique label $L_\phi(\{t_0, z\}) = \{6\}$. The addition of the vertex t_0 and the labels of the (dashed) edges incident to t_0 are illustrated in Figure 4.7a. Denote the vertex sets $A = \{s^{x_i}, u_p^{x_i}, v_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$, $B = \{w_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$, and $C = \{t_p^{x_i} : 1 \leq i \leq n,\ 1 \leq p \leq 3\}$. Note that $V_\phi = A \cup B \cup C \cup \{t_0\}$. This completes the construction of the graph G_ϕ and its labelling L_ϕ.

For every $i \in \{1, 2, \ldots, n\}$ the graph $G_{\phi,i}$ has 16 vertices. Furthermore, for every $p \in \{1, 2, 3\}$, the 4 vertices of the pth branch of $G_{\phi,i}$ also belong to a branch of $G_{\phi,j}$, for some $j \neq i$. Therefore, together with the vertex t_0, the graph G_ϕ has in total $10n + 1$ vertices. We now present the auxiliary lemmas 4.15-4.17 which are necessary for the proof of Theorem 4.18. More specifically, Lemma 4.15 gives



(a) The addition of vertex t_0. There exists in G_ϕ also the edge $\{t_0, w_0^{x_n}\}$ with label $L_\phi(\{t_0, w_0^{x_n}\}) = \{5\}$.



(b) The gadget for the clause $(x_i \oplus x_j)$.

FIGURE 4.7: Construction of G_ϕ and L_ϕ.

the total number of labels in L_ϕ, Lemma 4.16 proves that L_ϕ is TC satisfying on G_ϕ and Lemma 4.17 proves a set of statements for all TC satisfying labellings on G_ϕ. We can then use those statements in Theorem 4.18, when considering a minimal sub-labelling L of L_ϕ on G_ϕ, to show a lower bound on the number of labels contained in L (and, consequently, an upper bound on the number of labels that can be removed by L_ϕ without violating TC).

Lemma 4.15. *The labelling L_ϕ assigns $\frac{17}{4}n^2 + 28n + 1$ labels to the edges of G_ϕ.*

Proof. The vertex t_0 has in total 3 incident edges (to vertices $s^{x_i}, u_0^{x_i}, v_0^{x_i}$) to every base of a variable x_i of ϕ, 3 incident edges (to vertices $t_p^{x_i}, u_p^{x_i}, v_p^{x_i}$, where $1 \le p \le 3$) to every clause $(x_i \oplus x_j)$ of ϕ (i.e., to one branch to x_i and one branch of x_j), and one incident edge to vertex $w_0^{x_n}$. That is, t_0 has in total $3n+3m+1 = 3n+3\cdot\frac{3}{2}n+1 = \frac{15}{2}n+1$ incident edges, each of them having one label in L_ϕ.

Furthermore there exist in total $\frac{m(m-1)}{2}$ edges among the vertices $\{t_p^{x_i} : 1 \le i \le n,\ 1 \le p \le 3\}$, as well as $\frac{(n+m)(n+m-1)}{2}$ edges among the vertices $\{w_p^{x_i} : 1 \le i \le n,\ 0 \le p \le 3\}$, each of them having one label in L_ϕ. Therefore, since $m = \frac{3}{2}n$, L_ϕ assigns in total $\frac{17}{4}n^2 - 2n$ labels for these edges.

Moreover, the labelling L_ϕ assigns to every variable x_i of ϕ in total 12 labels, i.e., two labels for each of the transition edges $\{u_0^{x_i}, w_0^{x_i}\}$, $\{w_0^{x_i}, v_0^{x_i}\}$ and one label for each of the edges $\{\{s^{x_i}, u_0^{x_i}\}, \{s^{x_i}, v_0^{x_i}\}, \{u_0^{x_i}, u_p^{x_i}\}, \{v_0^{x_i}, v_p^{x_i}\} : 1 \leq p \leq 3\}$.

Finally, L_ϕ assigns to every clause $(x_i \oplus x_j)$ of ϕ in total 7 labels, i.e., two labels for each of the transition edges $\{u_p^{x_i}, w_p^{x_i}\}$, $\{w_p^{x_i}, v_p^{x_i}\}$ and one label for each of the edges $\{u_p^{x_i}, t_p^{x_i}\}$, $\{v_p^{x_i}, t_p^{x_i}\}$, $\{t_p^{x_i}, w_p^{x_i}\}$, where x_i is associated with the pth branch of $G_{\phi,i}$. That is, L_ϕ assigns in total $7m = \frac{21}{2}n$ labels for all clauses of ϕ.

Summarizing, the labelling L_ϕ assigns to the edges of the graph G_ϕ a total of $\left(\frac{15}{2}n + 1\right) + \left(\frac{17}{4}n^2 - 2n\right) + 12n + \frac{21}{2}n = \frac{17}{4}n^2 + 28n + 1$ labels. ■

Lemma 4.16. *The labelling L_ϕ satisfies TC on G_ϕ.*

Proof. We will prove that there exists a temporal path in L_ϕ between any pair of vertices of $V_\phi = A \cup B \cup C \cup \{t_0\}$.

For any two vertices $b, b' \in B$ there exists a temporal path from b to b' and from b' to b, due to the edge $\{b, b'\}$ with label 7. Similarly, for any two vertices $c, c' \in C$ there exists a temporal path from c to c' and from c' to c, due to the edge $\{c, c'\}$ with label 7. Let $a_1, a_2 \in A$. There exists a temporal path from a_1 to a_2 as follows: start from a_1, follow $P_{i,p}$ (or $Q_{i,p}$) upwards until $t_p^{x_i}$ with greatest label 4, then go to t_0 with label 5, and finally from t_0 to a_2 with label 6. In the special case where a_1 and a_2 lie on the same path $P_{i,p}$ (resp. $Q_{i,p}$) and a_1 appears before a_2 in $P_{i,p}$ (resp. $Q_{i,p}$), there exists clearly a temporal path from a_1 to a_2 along $P_{i,p}$ (resp. $Q_{i,p}$).

Let $a \in A$ and $b \in B$. Note that $b = w_p^{x_i}$ for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{0, 1, 2, 3\}$. There exists the temporal path from b to a as follows. First follow the edge $\{w_p^{x_i}, u_p^{x_i}\}$ (with label 1), then follow upwards the path $P_{i,p}$ until one of the vertices $\{t_1^{x_i}, t_2^{x_i}, t_3^{x_i}\}$ (with maximum label 4), then go to t_0 with label 5 and finally reach a with label 6. Furthermore there exists the temporal path from a to b as follows. Assume first that $a = s^{x_i}$, for some $i \in \{1, 2, \ldots, n\}$. If $b = w_0^{x_i}$ then there exists the temporal path on the edges $\{s^{x_i}, u_0^{x_i}\}$ (with label 1) and $\{u_0^{x_i}, w_0^{x_i}\}$ (with label 2). If $b \neq w_0^{x_i}$ then there exists the temporal path from s^{x_i} to $w_0^{x_i}$ (with maximum label 2), followed by the edge $\{w_0^{x_i}, b\}$ (with label 7). Assume now that $a \neq s^{x_i}$, for every $i \in \{1, 2, \ldots, n\}$. That is, $a = u_p^{x_i}$ or $a = v_p^{x_i}$, for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{0, 1, 2, 3\}$. If $b = w_p^{x_i}$ then there exists the temporal path from a to b on the edge $\{a, b\}$ (with label 1). If $b \neq w_p^{x_i}$ then there exists the temporal path from a to b through the edges $\{a, w_p^{x_i}\}$ (with label 1) and $\{w_p^{x_i}, b\}$ (with label 7). That is, there exists a temporal path in L_ϕ between any $a \in A$ and any $b \in B$.

Let $b \in B$, i.e., $b = w_p^{x_i}$ for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{0, 1, 2, 3\}$. Then there exists a temporal path from b to every vertex $c \in C$ as follows. If $p = 0$ then start with the edge $\{w_0^{x_i}, u_0^{x_i}\}$ (of label 1), continue upwards with a temporal path (of maximum label 4) until $t_1^{x_i} \in C$ and then continue to any other vertex $c \in C$ with the edge $\{t_1^{x_i}, c\}$ (of label 7). If $p \in \{1, 2, 3\}$ then reach $t_p^{x_i} \in C$ with the edge $\{w_p^{x_i}, t_p^{x_i}\}$ (of label 1) and continue to any other vertex $c \in C$ with the edge $\{t_p^{x_i}, c\}$ (of label 7). That is, there exists a temporal path from any $b \in B$ to any vertex of the set C. Now let $c \in C$, i.e., $c = t_p^{x_i}$ for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{1, 2, 3\}$. Then there exists a temporal path from c to every vertex $b \in B$ as follows. First reach the vertex $w_p^{x_i} \in B$ with the edge $\{t_p^{x_i}, w_p^{x_i}\}$ (of label 1) and then continue to any other vertex $c \in C$ with the edge $\{w_p^{x_i}, c\}$ (of label 7). That is, there exists a temporal path in L_ϕ between any $b \in B$ and any $c \in C$.

Let $a \in A$, i.e., $a \in \{s^{x_i}, u_p^{x_i}, v_p^{x_i}\}$ for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{0, 1, 2, 3\}$. Then there exists at least one path from a upwards to a vertex $c \in \{t_1^{x_i}, t_2^{x_i}, t_3^{x_i}\}$ (with maximum label 4). Once we have (temporally) reached c from a, we can (temporally) continue to any other $c' \in C$ through the edge $\{c, c'\}$ (of label 7). That is, there exists a temporal path from any $a \in A$ to any vertex of C. Now let $c \in C$, i.e., $c = t_p^{x_i}$ for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{1, 2, 3\}$. Then there exists a temporal path from c to every vertex $a \in A$ as follows. First reach the vertex t_0 with the edge $\{t_p^{x_i}, t_0\}$ (of label 5) and then continue to any vertex $a \in A$ with the edge $\{t_0, a\}$ (of label 6). That is, there exists a temporal path in L_ϕ between any $a \in A$ and any $c \in C$.

Finally, there exists a temporal path between t_0 and every vertex of $A \cup C \cup \{w_0^{x_n}\}$, since t_0 is a neighbour with all these vertices. Let $b \in B$, i.e., $b = w_p^{x_i}$ for some $i \in \{1, 2, \ldots, n\}$ and some $p \in \{0, 1, 2, 3\}$. Then there exists a temporal path from $w_p^{x_i}$ to t_0 with the edges $\{w_p^{x_i}, u_p^{x_i}\}$ (with label 1) and $\{u_p^{x_i}, t_0\}$ (with label 6). On the other hand, there exists a temporal path from t_0 to every vertex $b = w_p^{x_i} \in B$, as follows. First reach the vertex $w_0^{x_n}$ with the edge $\{t_0, w_0^{x_n}\}$ (of label 5) and then, if $b \neq w_0^{x_n}$, continue with the edge $\{w_0^{x_n}, b\}$ (of label 7). That is, there exists a temporal path in L_ϕ between t_0 and any vertex in $A \cup B \cup C$.

Summarizing, there exists a temporal path between any pair of vertices of $V_\phi = A \cup B \cup C \cup \{t_0\}$, i.e., the labelling L_ϕ satisfies TC on G_ϕ. ∎

Lemma 4.17. *Let $L \subseteq L_\phi$ be a labelling of the graph G_ϕ. If L satisfies TC on G_ϕ, then L contains:*

(a) at least one label for every transition edge $\{u_p^{x_i}, w_p^{x_i}\}$ and $\{w_p^{x_i}, v_p^{x_i}\}$, where $i \in \{1, 2, \ldots, n\}$ and $p \in \{0, 1, 2, 3\}$,

(b) the label of each edge $\{t_p^{x_i}, w_p^{x_i}\}$, where $i \in \{1, 2, \ldots, n\}$ and $p \in \{1, 2, 3\}$,

(c) the labels of all edges of G_ϕ among the vertices $\{t_p^{x_i} : 1 \leq i \leq n,\ 1 \leq p \leq 3\}$,

(d) the labels of all edges among the vertices $\{w_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$,

(e) the label of each edge incident to t_0, and

(f) the labels of all edges of the path $P_{i,p}$ or the labels of all edges of the path $Q_{i,p}$, where $i \in \{1, 2, \ldots, n\}$ and $p \in \{1, 2, 3\}$.

Proof. (a) First assume that L does not keep any time label on the transition edge $\{u_p^{x_i}, w_p^{x_i}\}$ (resp. $\{w_p^{x_i}, v_p^{x_i}\}$), where $i \in \{1, 2, \ldots, n\}$ and $p \in \{0, 1, 2, 3\}$. Then there does not exist in L any temporal path from $u_p^{x_i}$ (resp. $v_p^{x_i}$) to $w_p^{x_i}$, even if L maintains all other edge labels from L_ϕ. This is a contradiction. Therefore L keeps at least one label on the transition edge $\{u_p^{x_i}, w_p^{x_i}\}$ (resp. $\{w_p^{x_i}, v_p^{x_i}\}$).

(b) Now assume that L does not contain the label of some edge $\{t_p^{x_i}, w_p^{x_i}\}$, where $i \in \{1, 2, \ldots, n\}$ and $p \in \{1, 2, 3\}$. Then there does not exist in L any temporal path from $t_p^{x_i}$ to any vertex $w_q^{x_j} \in B$, even if L maintains all other edge labels from L_ϕ. This is a contradiction to the assumption that L satisfies TC on G_ϕ. Therefore L contains the label of each edge $\{t_p^{x_i}, w_p^{x_i}\}$, where $i \in \{1, 2, \ldots, n\}$ and $p \in \{1, 2, 3\}$.

(c) Consider two vertices $t_p^{x_i} \neq t_q^{x_j}$, $1 \leq i < j \leq n$, $1 \leq p, q \leq 3$. If L does not contain the label of the edge $\{t_p^{x_i}, t_q^{x_j}\}$, then there does not exist in L any temporal path from $t_p^{x_i}$ to $t_q^{x_j}$, which is a contradiction. Therefore L contains the labels of all edges of G_ϕ among the vertices $\{t_p^{x_i} : 1 \leq i \leq n,\ 1 \leq p \leq 3\}$.

(d) Assume that L does not contain the label of the edge $\{w_p^{x_i}, w_q^{x_j}\}$, for some $i, j \in \{1, 2, \ldots, n\}$ and $p, q \in \{0, 1, 2, 3\}$. Then there does not exist in L any temporal path from $w_p^{x_i}$ to $w_q^{x_j}$, which is a contradiction. Therefore L contains the labels of all edges among the vertices $\{w_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$.

(e) We now prove that L contains the label of each edge incident to t_0. Recall that the neighbours of t_0 in G_ϕ are exactly the vertices of the set $A \cup C \cup \{w_0^{x_n}\}$. Assume L does not have the label of the edge $e = \{t_0, w_0^{x_n}\}$. Then there exists no temporal path in L from t_0 to any vertex $w_p^{x_i} \in B$, even if L maintains all other edge labels from L_ϕ. This is a contradiction to the assumption that L satisfies TC on G_ϕ. Now assume that there exists a vertex $a \in A = \{s^{x_i}, u_p^{x_i}, v_p^{x_i} : 1 \leq i \leq n,\ 0 \leq p \leq 3\}$ such that L does not have the label of the edge $e = \{t_0, a\}$. Then there does not exist in L any temporal path from vertex t_0 to vertex a, which is again a contradiction. Finally assume that there exists a vertex $t_p^{x_i} \in C$, such that L does not have the label of the edge $e = \{t_0, t_p^{x_i}\}$. Then there does not exist in L any temporal path from vertex $u_p^{x_i}$ to vertex s^{x_i}, which is a contradiction. Therefore L contains the label of each edge incident to t_0.

(f) Assume that L misses from L_ϕ at least one label of the path $P_{i,p}$ and at least one label of the path $Q_{i,p}$, for some $i \in \{1, 2, \ldots, n\}$ and $p \in \{1, 2, 3\}$. Then there does not exist any temporal path from s^{x_i} to $t_p^{x_i}$, which is a contradiction. Therefore L contains the labels of all edges of the path $P_{i,p}$ or the labels of all edges of the path $Q_{i,p}$, where $i \in \{1, 2, \ldots, n\}$ and $p \in \{1, 2, 3\}$.

■

We are now ready to provide the proof of Theorem 4.18.

Theorem 4.18. *There exists a truth assignment τ of ϕ with $|\tau(\phi)| \geq k$ if and only if there exists a TC satisfying labelling $L \subseteq L_\phi$ of G_ϕ such that $|L_\phi \setminus L| \geq 9n + k$.*

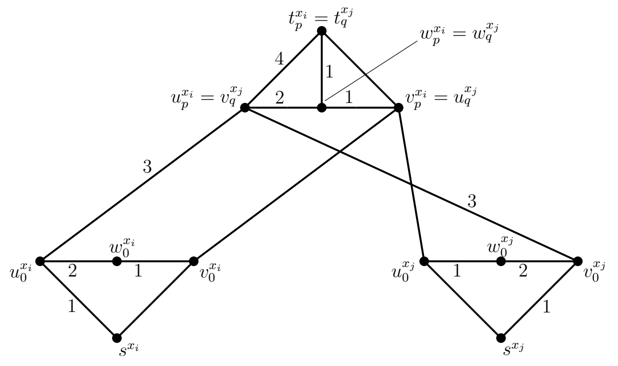
Proof. ($\Rightarrow$) Assume that there is a truth assignment τ that XOR-satisfies k clauses of ϕ. We construct a labelling L of G_ϕ by removing $9n+k$ labels from L_ϕ, as follows. First we keep in L all labels of L_ϕ on the edges incident to t_0. Furthermore we keep in L the label $\{7\}$ of all the edges $\{t_p^{x_i}, t_q^{x_j}\}$ and the label $\{7\}$ of all the edges $w_p^{x_i} w_q^{x_j}$. Moreover we keep in L the label $\{1\}$ of all the edges $\{t_p^{x_i}, w_p^{x_i}\}$. Let now

$i = 1, 2, \ldots, n$. If $x_i = 0$ in τ, we keep in L the labels of the edges of the paths $P_{i,1}, P_{i,2}, P_{i,3}$, as well as the label 1 of the edge $\{v_0^{x_i}, w_0^{x_i}\}$ and the label 2 of the edge $\{w_0^{x_i}, u_0^{x_i}\}$. Otherwise, if $x_i = 1$ in τ, we keep in L the labels of the edges of the paths $Q_{i,1}, Q_{i,2}, Q_{i,3}$, as well as the label 1 of the edge $\{u_0^{x_i}, w_0^{x_i}\}$ and the label 2 of the edge $\{w_0^{x_i}, v_0^{x_i}\}$.

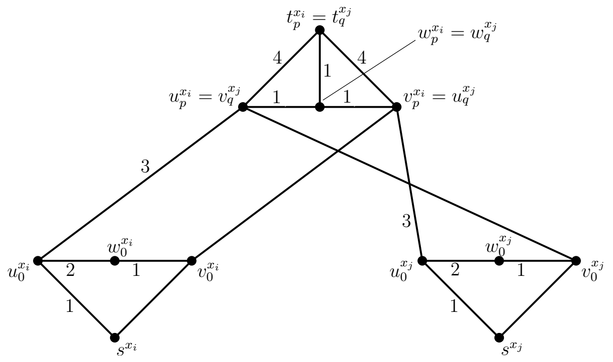
We now continue the labelling L as follows. Consider an arbitrary clause $\alpha = (x_i \oplus x_j)$ of ϕ. Assume that the variable x_i (resp. x_j) of the clause α corresponds to the pth (resp. to the qth) appearance of variable x_i (resp. x_j) in ϕ. Then, by the construction of G_ϕ, the pth branch of $G_{\phi,i}$ coincides with the qth branch of $G_{\phi,j}$, i.e., $u_p^{x_i} = v_q^{x_j}$, $v_p^{x_i} = u_q^{x_j}$, $w_p^{x_i} = w_q^{x_j}$, and $t_p^{x_i} = t_q^{x_j}$ (see Figure 4.7b). Let α be XOR-satisfied in τ, i.e., $x_i = \overline{x_j}$. If $x_i = \overline{x_j} = 0$ (i.e., $x_i = 0$ and $x_j = 1$) then we keep in L the label 1 of the edge $\{v_p^{x_i}, w_p^{x_i}\}$ and the label 2 of the edge $\{w_p^{x_i}, u_p^{x_i}\}$, see Figure 4.8a. In the symmetric case, where $x_i = \overline{x_j} = 1$ (i.e., $x_i = 1$ and $x_j = 0$), we keep in L the label 1 of the edge $\{u_p^{x_i}, w_p^{x_i}\}$ and the label 2 of the edge $\{w_p^{x_i}, v_p^{x_i}\}$. Let now α be XOR-unsatisfied in τ, i.e., $x_i = x_j$. Then, in both cases where $x_i = x_j = 0$ and $x_i = x_j = 1$, we keep in L the label 1 of both edges $\{v_p^{x_i}, w_p^{x_i}\}$ and $\{w_p^{x_i}, u_p^{x_i}\}$, see Figure 4.8b. This finalizes the labelling L of G_ϕ. It is easy to check that L satisfies TC on G_ϕ.

Summarizing, the labelling L misses in total 6 labels of L_ϕ for the edges $\{\{s^{x_i}, u_0^{x_i}\},\ \{s^{x_i}, v_0^{x_i}\},\ \{u_0^{x_i}, w_0^{x_i}\},\ \{w_0^{x_i}, v_0^{x_i}\},\ \{u_0^{x_i}, u_p^{x_i}\},\ \{v_0^{x_i}, v_p^{x_i}\} : 1 \le p \le 3,\ i = 1, 2, \ldots, n\}$. That is, L misses in total $6n$ labels of L_ϕ for all variables $x_1, x_2, \ldots, x_n$. For each of the k XOR-satisfied clauses $(x_i \oplus x_j)$ of ϕ, the labelling L misses in total 3 labels of L_ϕ for the edges $\{u_p^{x_i}, w_p^{x_i}\}$, $\{w_p^{x_i}, v_p^{x_i}\}$, $\{u_p^{x_i}, t_p^{x_i}\}$, $\{v_p^{x_i}, t_p^{x_i}\}$, $\{t_p^{x_i}, w_p^{x_i}\}$, where x_i is associated with the pth branch of $G_{\phi,i}$. That is, L misses in total $3k$ labels of L_ϕ for all XOR-satisfied clauses. Furthermore, for each of the $m-k$ XOR-satisfied clauses $(x_i \oplus x_j)$ of ϕ, the labelling L misses in total 2 labels of L_ϕ for the edges $\{u_p^{x_i}, w_p^{x_i}\}$, $\{w_p^{x_i}, v_p^{x_i}\}$, $\{u_p^{x_i}, t_p^{x_i}\}$, $\{v_p^{x_i}, t_p^{x_i}\}$, $\{t_p^{x_i}, w_p^{x_i}\}$, where x_i is associated with the pth branch of $G_{\phi,i}$. That is, L misses in total $2(m-k) = 3n - 2k$ labels of L_ϕ for all XOR-satisfied clauses. All other labels of L_ϕ remain in the labelling $L \subseteq L_\phi$. Therefore, L misses in total $6n + 3k + 3n - 2k = 9n + k$ labels from L_ϕ.

($\Leftarrow$) Assume that $r(G_\phi, L_\phi) \ge 9n+k$ and let $L \subseteq L_\phi$ be a TC preserving labelling of G_ϕ with $|L_\phi \backslash L| = r(G_\phi, L_\phi) \ge 9n+k$, i.e., $G_\phi(L)$ is minimal. Let $i \in \{1, 2, \ldots, n\}$. For every $p \in \{1, 2, 3\}$, L contains by Lemma 4.17(f) the labels of all edges of the path $P_{i,p}$ *or* the labels of all edges of the path $Q_{i,p}$. Therefore, there exist at least two indices $p_1, p_2 \in \{1, 2, 3\}$ such that L contains the labels of all edges of the



(a) Case $x_i = \overline{x_j} = 0$.



(b) Case $x_i = x_j = 0$.

FIGURE 4.8: The labelling $L \subseteq L_\phi$ of the edges of Fig. 4.7b for the clause $\alpha = (x_i \oplus x_j)$ of ϕ.

paths P_{i,p_1}, P_{i,p_2} *or* the labels of all edges of the paths Q_{i,p_1}, Q_{i,p_2}. Without loss of generality let $p_1 = 1$ and $p_2 = 2$ and let L contain the labels of all edges of the paths $P_{i,1}, P_{i,2}$ (the other cases can be dealt with in the same way by symmetry). Assume that L also contains the labels of all edges of the path $Q_{i,3} = (s^{x_i}, v_0^{x_i}, v_3^{x_i}, t_3^{x_i})$. Then we can modify the labelling L to a labelling L' as follows. First remove from L the labels of the edges $\{s^{x_i}, v_0^{x_i}\}$ and $\{v_0^{x_i}, v_3^{x_i}\}$ and add instead the labels of the edges $\{u_0^{x_i}, u_3^{x_i}\}$ and $\{u_3^{x_i}, t_3^{x_i}\}$ (if they do not exist yet in L). Furthermore change the labels of the transition edges $\{v_0^{x_i}, w_0^{x_i}\}$ and $\{w_0^{x_i}, u_0^{x_i}\}$ to the labels 1 and 2, respectively. Note that in the resulting labelling L', both edges $\{u_3^{x_i}, t_3^{x_i}\}$ and $\{v_3^{x_i}, t_3^{x_i}\}$ are labelled. Furthermore $L' \subseteq L_\phi$ and L' does not have more labels than L, and thus $|L_\phi \setminus L'| \geq |L_\phi \setminus L| = r(G_\phi, L_\phi)$. Moreover, it is easy to check that L' still satisfies TC on G_ϕ, as L satisfies TC as well. So, it must also be $|L_\phi \setminus L'| = r(G_\phi, L_\phi)$, i.e., $G_\phi(L')$ is also minimal. Therefore, we may assume without loss of generality that for any minimal labelling $L \subseteq L_\phi$, L contains the labels of all edges of the paths $P_{i,1}, P_{i,2}, P_{i,3}$ *or* the labels of all edges of the paths $Q_{i,1}, Q_{i,2}, Q_{i,3}$.

From Lemma 4.17(a), L contains at least $2n + 2m$ labels on the edges of the form $\{u_p^{x_i}, w_p^{x_i}\}$ or $\{w_p^{x_i}, v_p^{x_i}\}$, since there are exactly $2n$ transition edges on the different bases of G_ϕ and $2m$ transition edges on the different branches of G_ϕ. From Lemma 4.17(b), L contains m additional labels, one for each branch, more specifically for the respective edge $\{t_p^{x_i}, w_p^{x_i}\}$ of the branch. From Lemma 4.17(c), L contains $\frac{m(m-1)}{2}$ extra labels among the vertices $\{t_p^{x_i} : 1 \leq i \leq n, 1 \leq p \leq 3\}$. From Lemma 4.17(d), L also contains $\frac{(n+m)(n+m-1)}{2}$ additional labels among the vertices $\{w_p^{x_i} : 1 \leq i \leq n, 0 \leq p \leq 3\}$. From Lemma 4.17(e), L also contains $\frac{15}{2}n+1$ labels on the edges incident to t_0. Finally, from Lemma 4.17(f), L contains at least $4n+m$ additional labels: for each $G_{\phi,i}$, L contains at least 4 labels, namely one label on the base edge $\{s^{x_i}, u_0^{x_i}\}$ or on the base edge $\{s^{x_i}, v_0^{x_i}\}$ and, for every $p \in \{1,2,3\}$, one label on the edge $\{u_0^{x_i}, u_p^{x_i}\}$ or on the edge $\{v_0^{x_i}, v_p^{x_i}\}$; also, for each branch of G_ϕ, L contains at least 1 label, namely a label on the edge $\{u_p^{x_j}, t_p^{x_j}\}$ or on the edge $\{v_p^{x_j}, t_p^{x_j}\}$, for some $p \in \{1,2,3\}$ and $j \in \{1,2,\ldots,m\}$.

Notice that all the labels of L mentioned above are on different edges, so no subset of labels has been accounted for more than once. Therefore, since $m = \frac{3n}{2}$, L contains at least:

$$c(L) \geq \frac{17}{4}n^2 + 17n + \frac{n}{2} + 1 \tag{4.1}$$

labels.

Now we construct from the labelling $L \subseteq L_\phi$ a truth assignment τ for the formula ϕ as follows. For every $i \in \{1,2,\ldots,n\}$, if L contains the labels of all edges of the paths $P_{i,1}, P_{i,2}, P_{i,3}$, then we define $x_i = 0$ in τ. Otherwise, if L contains the labels of all edges of the paths $Q_{i,1}, Q_{i,2}, Q_{i,3}$, then we define $x_i = 1$ in τ. We will prove that $|\tau(\phi)| \geq k$, i.e., that τ XOR-satisfies at least k clauses of the formula ϕ.

Let $\alpha = (x_i \oplus x_j)$, where $i, j \in \{1,2,\ldots,n\}$, be a clause of ϕ that is *not* XOR-satisfied by τ in ϕ. Let x_i (resp. x_j) be associated with the pth (resp. qth) branch of $G_{\phi,i}$ (resp. of $G_{\phi,j}$). Since α is *not* XOR-satisfied, either $x_i = x_j = 0$ or $x_i = x_j = 1$ in τ. If $x_i = x_j = 0$ in τ, it follows by the definition of the assignment τ that the labelling L contains the labels of all edges of the path $P_{i,p}$ *and* of the path $P_{j,q}$. Therefore, the p^{th} branch of $G_{\phi,i}$, which is identified with the q^{th} branch of $G_{\phi,j}$, has both edges $\{t_p^{x_i}, u_p^{x_i}\} \equiv \{t_q^{x_j}, v_q^{x_j}\}$ and $\{t_p^{x_i}, v_p^{x_i}\} \equiv \{t_q^{x_j}, u_q^{x_j}\}$ labelled under L, with one label each. The same holds if $x_i = x_j = 1$, where all edges of both paths $Q_{i,p}$ *and* $Q_{j,q}$ are labelled. So, for all the branches of G_ϕ that correspond to non-satisfied clauses of ϕ by the truth assignment τ, L contains an additional label (to the ones accounted for by using the result of Lemma 4.17(f)). The number of clauses that are not satisfied by τ in ϕ is exactly $m - |\tau(\phi)| = \frac{3}{2}n - |\tau(\phi)|$.

Thus, it follows by Equation (4.1), by adding the extra $\frac{3}{2}n - |\tau(\phi)|$, that L contains in total at least:

$$\begin{aligned} c(L) &\geq \frac{17}{4}n^2 + 17n + \frac{n}{2} + 1 + (\frac{3n}{2} - |\tau(\phi)|) \\ &= \frac{17}{4}n^2 + 19n + 1 - |\tau(\phi)| \end{aligned}$$

labels.

Recall now that we have already shown in Lemma 4.15 that L_ϕ has a total of $\frac{17}{4}n^2 + 28n + 1$ labels. Therefore, we have:

$$|L_\phi \setminus L| = c(L_\phi) - c(L) \leq 9n + |\tau(\phi)|.$$

However, by our initial assumption:

$$|L_\phi \setminus L| = r(G_\phi, L_\phi) \geq 9n + k.$$

Therefore $9n + k \leq |L_\phi \setminus L| \leq 9n + |\tau(\phi)|$, and thus $|\tau(\phi)| \geq k$, i.e., the truth assignment τ satisfies at least k clauses of ϕ. This completes the proof of the theorem. ■

The next corollary follows immediately by Theorem 4.18.

Corollary 4.19. *Let $OPT_{mon\text{-}Max\text{-}XOR(3)}(\phi)$ the greatest number of clauses that can be simultaneously XOR-satisfied by a truth assignment of ϕ. Then $r(G_\phi, L_\phi) = 9n + OPT_{mon\text{-}Max\text{-}XOR(3)}(\phi)$.*

Proof. Let τ be a truth assignment that satisfies $k = \mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi)$ clauses of ϕ. Then there exists by Theorem 4.18 a TC satisfying labelling $L \subseteq L_\phi$ of G_ϕ such that $|L_\phi \setminus L| \geq 9n + k$. Thus, since $r(G_\phi, L_\phi) \geq |L_\phi \setminus L|$, it follows that $r(G_\phi, L_\phi) \geq 9n + \mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi)$. Conversely, let $L \subseteq L_\phi$ be a labelling of G_ϕ such that $|L_\phi \setminus L| = r(G_\phi, L_\phi)$. Then there exists by Theorem 4.18 a truth assignment τ that satisfies at least $r(G_\phi, L_\phi) - 9n$ clauses of ϕ. Thus $\mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi) \geq r(G_\phi, L_\phi) - 9n$, which completes the proof. ■

Using Theorem 4.18 and Corollary 4.19, we are now ready to prove the main theorem of this section.

Theorem 4.20. *The problem of computing $r(G, L)$ on an undirected temporally connected graph $G(L)$ is APX-hard.*

Proof. Denote by $\mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi)$ the greatest number of clauses that can be simultaneously XOR-satisfied by a truth assignment of ϕ. The proof is done by an *L-reduction* [91] from the monotone Max-XOR(3) problem, i.e. by an approximation preserving reduction which linearly preserves approximability features. For such a reduction, it suffices to provide a polynomial-time computable function g and two constants $\alpha, \beta > 0$ such that:

- $r(G_\phi, L_\phi) \leq \alpha \cdot \mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi)$, for any monotone XOR(3) formula ϕ, and
- for any TC satisfying labelling $L \subseteq L_\phi$ of G_ϕ, $g(L)$ is a truth assignment for ϕ and $\mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi) - |g(L)| \leq \beta \cdot (r(G_\phi, L_\phi) - |L_\phi \setminus L|)$, where $|g(L)|$ is the number of clauses of ϕ that are satisfied by $g(L)$.

We will prove the first condition for $\alpha = 13$. Note that a random truth assignment XOR-satisfies each clause of ϕ with probability $\frac{1}{2}$, and thus there exists an assignment τ that XOR-satisfies at least $\frac{m}{2}$ clauses of ϕ. Therefore $\mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi) \geq \frac{m}{2} = \frac{3}{4}n$, and thus $n \leq \frac{4}{3}\mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi)$. Now Corollary 4.19 implies that:

$$\begin{aligned} r(G_\phi, L_\phi) &= 9n + \mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi) \\ &\leq 9 \cdot \frac{4}{3}\mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi) + \mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi) \\ &= 13 \cdot \mathrm{OPT}_{\text{mon-Max-XOR(3)}}(\phi). \end{aligned} \tag{4.2}$$

To prove the second condition for $\beta = 1$, consider an arbitrary labelling $L \subseteq L_\phi$ of G_ϕ. As described in the ($\Leftarrow$)-part of the proof of Theorem 4.18, we construct in polynomial time a truth assignment $g(L) = \tau$ that satisfies at least $|L_\phi \setminus L| - 9n$ clauses of ϕ, i.e. $|g(L)| = |\tau(\phi)| \geq |L_\phi \setminus L| - 9n$. Then:

$$\begin{aligned} OPT_{\text{mon-Max-XOR(3)}}(\phi) - |g(L)| &\leq OPT_{\text{mon-Max-XOR(3)}}(\phi) - |L_\phi \setminus L| + 9n \\ &= r(G_\phi, L_\phi) - 9n - |L_\phi \setminus L| + 9n \end{aligned} \tag{4.3}$$

$$= \; r(G_\phi, L_\phi) - |L_\phi \setminus L|.$$

This completes the proof of the Theorem. ■

Note. In fact, we have also shown (Theorem 4.18) that the problem of computing the removal profit is NP-hard in the strong sense, since all numbers used in the reduction are constant integers.

4.4.3 Temporally connected random labellings have high removal profit

In this section, we show that dense graphs with random labels have the property TC and have a very high removal profit asymptotically almost surely. More specifically, we consider the complete graph and the Erdös-Renyi model of random graphs, $G_{n,p}$ and we examine whether we can delete labels from such temporal graphs and continue preserving TC.

The (single-labelled) model of temporal graphs that we consider here is that of *uniform random temporal graphs* [8]. This is a model that we also considered in Chapter 3, Section 3.4.1, for the purpose of analysing the journey decomposition of temporal flows in *directed* temporal networks; here, however, we consider *undirected* graphs.

Definition 4.21. [8] A *uniform random temporal graph* is a graph G on n vertices, $n \in \mathbb{N}$, each edge of which receives exactly one label uniformly at random from a set $\{1, 2, \ldots, \alpha\}$, $\alpha \in \mathbb{N}$ and the selection of the label of an edge is independent from the selection of the label of any other edge.

4.4.3.1 High removal profit in the complete graph

Theorem 4.22. *In the uniform random temporal graph where the underlying graph G is the complete graph (clique) of n vertices and $\alpha \geq 4$, we can delete all but $\Theta(n \log n + \log^2 n)$ labels without violating TC, with probability at least $1 - \frac{1}{n^2}$.*

Proof. First, note that any set $\{1, 2, \ldots, \alpha\}$ of α consecutive natural numbers can be partitioned into 4 disjoint almost equal subsets of consecutive numbers, A_1, A_2, A_3, A_4. Indeed, let $\alpha = 4k + v$, where $k \in \mathbb{N}$ and $v \in \{1, 2, 3, 4\}$.

For $v = 0$, we use $A_1 = \{1, \dots, k\}, A_2 = \{k+1, \dots, 2k\}, A_3 = \{2k+1, \dots, 3k\}, A_4 = \{3k+1, \dots, 4k\}$.

For $v = 1$, we use $A_1 = \{1, \dots, k\}, A_2 = \{k+1, \dots, 2k\}, A_3 = \{2k+1, \dots, 3k\}, A_4 = \{3k+1, \dots, 4k+1\}$.

For $v = 2$, we use $A_1 = \{1, \dots, k\}, A_2 = \{k+1, \dots, 2k\}, A_3 = \{2k+1, \dots, 3k+1\}, A_4 = \{3k+2, \dots, 4k+2\}$.

For $v = 3$, we use $A_1 = \{1, \dots, k\}, A_2 = \{k+1, \dots, 2k+1\}, A_3 = \{2k+2, \dots, 3k+2\}, A_4 = \{3k+3, \dots, 4k+3\}$.

In any of the above four cases, each particular edge of the clique K_n receives a single random label l, with:

$$Pr[l \in A_i] \geq \frac{k}{4k+3}, \ \forall i = 1, 2, 3, 4.$$

Since $k \geq 1$ (because $\alpha \geq 4$), we have $\frac{k}{4k+3} \geq \frac{1}{7}$. So, we get the following Lemma:

Lemma 4.23. *For each particular edge e of K_n and for the label l that it receives, it holds that $Pr[l \in A_i] \geq \frac{k}{4k+3}$, $\forall i = 1, 2, 3, 4$.*

Now, colour *green*(g), *yellow*(y), *blue*(b) and *red*(r) the edges that are assigned a label in A_1, A_2, A_3 and A_4 respectively.

Definition 4.24. A temporal router (see Figure 4.9) of a clique $G = K_n = (V, E)$ is a subgraph $R = (V_R, E_R)$ of G, with $2\gamma \log n + 1$ vertices, γ being a constant such that $\gamma \geq 4 \cdot \frac{1}{\log_2 \frac{2500}{2499}}$, with the following properties (all logarithms are with base 2 here):

a) V_R is the union of a particular vertex v_0 (called the *centre* of R) and two equisized vertex sets V_{in} and V_{out}, each of $\gamma \log n$ vertices, and

b) R is the induced subgraph of G formed from V_R (so it is a clique itself).

Note that R has $|E_R| = 2\gamma \log n + \frac{(2\gamma \log n) \cdot (2\gamma \log n - 1)}{2}$ edges.

Let w, w' be any two vertices of the clique that are not in V_R. We consider the edges connecting w to V_{in} and the edges connecting w' to V_{out}; using those edges and the edges of R, there are $\gamma \log n$ edge-disjoint paths of length 4 (each) connecting w and w'. Let us call those paths *special paths* and note that every such path

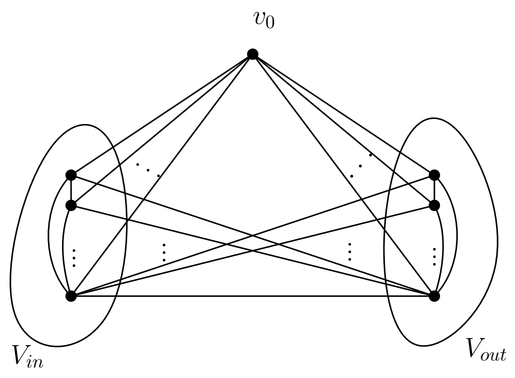


FIGURE 4.9: Temporal router of a clique

uses edges of the form $\{w, v_{in}\}, \{v_{in}, v_0\}, \{v_0, v_{out}\}, \{v_{out}, w'\}$, where $v_{in} \in V_{in}$ and $v_{out} \in V_{out}$ (see Figure 4.10).

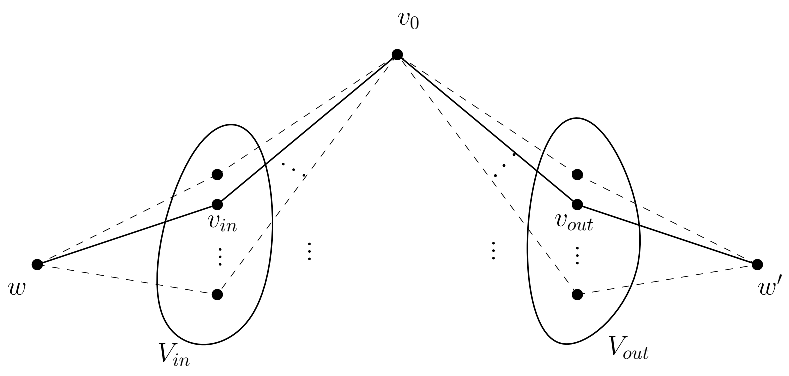


FIGURE 4.10: A special path connecting w and w'.

Each special path $P = (w, v_{in}, v_0, v_{out}, w')$ connecting w and w' becomes a $w \to w'$ journey if the label l_1 of $\{w, v_{in}\}$ is in A_1, the label l_2 of $\{v_{in}, v_0\}$ is in A_2, the label l_3 of $\{v_0, v_{out}\}$ is in A_3, and the label l_4 of $\{v_{out}, w'\}$ is in A_4. Then, the probability that P is a journey is at least $\left(\frac{1}{7}\right)^4$, due to independence of the labels' selection.

Since all special paths that connect w and w' are edge-disjoint, the probability that none of them is a $w \to w'$ journey is:

$$\begin{aligned} Pr[\text{no special path is a } w \to w' \text{ journey}] = \left(1 - \frac{1}{7^4}\right)^{\gamma \log n} &< \left(1 - \frac{1}{2500}\right)^{\gamma \log n} \\ &= n^{-\gamma \log \frac{2500}{2499}}. \end{aligned}$$

Therefore, we have:

Lemma 4.25. *For any two particular vertices w, w' of $V \setminus V_R$, the probability that there is a special path P from w to w' that is a $w \to w'$ journey is at least $1 - n^{-\gamma \log_2 \frac{2500}{2499}}$.*

Now, we consider only the edges and labels of R and, for each $w \in V \setminus V_R$, we consider only the edges connecting w to each vertex of R; the sparsified graph $G' = (V, E')$ has, thus, $|E'| = \frac{(2\gamma \log n+1)\cdot 2\gamma \log n}{2} + (n - (2\gamma \log n + 1))\cdot(2\gamma \log n+1) = \Theta(n \log n + \log^2 n)$ edges. We will show that we need only consider the edges (and labels) of G' to maintain TC in G, i.e., that G' itself is temporally connected, with probability at least $1 - \frac{1}{n^2}$.

Consider any pair, w, w', of vertices of the uniform random temporal graph on K_n and a temporal router R. Also, consider the graph G' as described above, with the labelling implied by the uniform random labelling on the clique. If $w, w' \in V_R$, then they are directly connected via a labelled edge in G' and thus a journey exists both ways between them. If $w \in V_R$ and $w \in V \setminus V_R$, then again there is a direct labelled edge in G' connecting w and w', so there is a journey between them either way.

It remains to examine the existence in G' of journeys between pairs of vertices w, w', $w \neq w'$, none of which is in V_R; there are at most n^2 such pairs of vertices. Under the random labelling on G, let $\mathcal{E}_1$ be the event that there exists a pair $w, w' \in V \setminus V_R$ such that there is no $w \to w'$ journey via a special path through R. Also, let $\mathcal{E}_2$ be the event that for a specific pair $w, w' \in V \setminus V_R$, there is no $w \to w'$ journey via a special path through R. Then,

$$Pr[\mathcal{E}_1] \leq n^2 Pr[\mathcal{E}_2] \text{ (by the Union Bound).}$$

So, we have:

$$Pr[G' \text{ is not temporally connected}] \leq n^2 n^{-\gamma \log_2 \frac{2500}{2499}}.$$

Note that Lemma 4.25 gives an upper bound on the probability of the event $\mathcal{E}_2$. Set γ to be $\gamma \geq 4 \cdot \frac{1}{\log_2 \frac{2500}{2499}}$. Then, we have:

$$Pr[G' \text{ is not temporally connected}] \leq n^{-2}.$$

■

4.4.3.2 High removal profit in dense random Erdös-Renyi graphs

In this section, we consider the underlying graph $G = (V, E)$ to be an instance of the Erdös-Renyi graph model, $G_{n,p}$, with $p \geq 7\left(\frac{\gamma \ln n}{n}\right)^{\frac{1}{7}}$, $\gamma \geq 24$.

Definition 4.26 (Erdös-Renyi graphs). An instance of $G_{n,p}$ is formed when for every pair of vertices u, v among a total number of n vertices, the edge $\{u, v\}$ is chosen to exist with probability p independently of any other edge.

We will also use the Multiplicative Chernoff bound, as described below:
Chernoff bound [11] Suppose $X_1, \ldots, X_n$ are independent random variables taking values in $\{0, 1\}$. Let X denote their sum and let $\mu = E[X]$ denote the sum's expected value. Then, for $0 < \delta < 1$:
$Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2\mu}{3}}$, and
$Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2\mu}{2}}$.

Notice that $G_{n,p}$ is almost surely connected for any $p \geq 2\frac{\ln n}{n}$ [24]. As in the previous section, we consider here a uniform random temporal graph on G, i.e., we consider each edge of G to receive exactly one label uniformly at random from a set $\{1, 2, \ldots, \alpha\}$, with $\alpha \geq 4$. The selection of the label of an edge is independent of the selection of the label of any other edge. Also, the label selection process is independent of the process of selection of edges in $G_{n,p}$. As in Theorem 4.22, we consider partitioning $\{1, 2, \ldots, \alpha\}$ into *four consecutive subsets,* A_1, A_2, A_3, A_4*, of consecutive positive integers*, where each subset is of size either $\lfloor \frac{\alpha}{4} \rfloor$ or $\lfloor \frac{\alpha}{4} \rfloor + 1$; such a partition is always possible. Now colour *green*(g), *yellow*(y), *blue*(b) and *red*(r) the edges that are assigned a label in A_1, A_2, A_3 and A_4, respectively. As in Lemma 4.23, we have:

Lemma 4.27. *For each particular edge of G and for the label l that it receives, it holds that $Prob[l \in A_i] \geq \frac{1}{7}$, $\forall i = 1, 2, 3, 4$.*

In such instances of $G_{n,p}$, we cannot assume the existence of cliques such as the clique of the temporal router used in the previous section. Indeed, even for very dense instances of $G_{n,p}$, with $p = \frac{1}{2}$, the largest clique is at most of size $2 \ln n$ [24].

In order to "sparsify" labelled instances G of $G_{n,p}$, by removing labels without violating TC, we need to guarantee the existence of much sparser routing subsets of G.

Definition 4.28. Given two vertices v_1, v_2 of $G_{n,p}$, a *temporal router, $R(v_1, v_2)$, in an instance I of $G_{n,p}$* is a subgraph of I that has vertices v_1, v_2 and additional vertices $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ so that:

- v_1 connects directly to each $a_i, b_i,\ i = 1, \ldots, k$,
- v_2 connects directly to each $a_i, b_i,\ i = 1, \ldots, k$,
- each pair a_i, b_i is directly connected, $i = 1, \ldots, k$,
- each edge $\{a_i, b_i\}$ receives a green label, $i = 1, \ldots, k$,
- each edge $\{a_i, v_1\}$ receives a yellow label, $i = 1, \ldots, k$,
- each edge $\{v_1, b_i\}$ receives a blue label, $i = 1, \ldots, k$,
- each edge $\{v_2, a_i\}$ receives a blue label, $i = 1, \ldots, k$,
- each edge $\{b_i, v_2\}$ receives a yellow label, $i = 1, \ldots, k$.

Figures 4.11 and 4.12 show a temporal router $R(v_1, v_2)$ for $k = 1$ and $k = 2$ respectively.

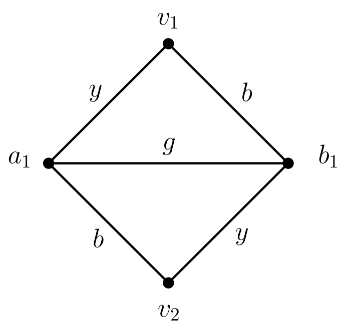


FIGURE 4.11: Temporal router $R(v_1, v_2)$ for $k = 1$.

Note. A temporal router $R(v_1, v_2)$ in an instance I of $G_{n,p}$ is *temporally connected* since:

- any a_i can reach any b_j, via a journey through v_1, i.e., (a_i, v_1, b_j) is a journey,
- any b_i can reach any a_j, via a journey through v_2, i.e., (b_i, v_2, a_j) is a journey,
- any a_i can reach any $a_j \neq a_i$, via a journey through b_i and then v_2, i.e., (a_i, b_i, v_2, a_j) is a journey,

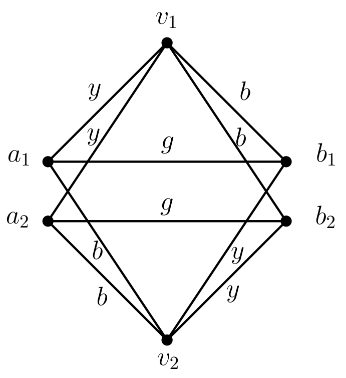


FIGURE 4.12: Temporal router $R(v_1, v_2)$ for $k = 2$.

- any b_i can reach any $b_j \neq b_i$, via a journey through a_i and then v_1, i.e., (b_i, a_i, v_1, b_j) is a journey,
- v_1 can reach v_2, via any a_i, i.e., (v_1, a_i, v_2) is a journey,
- v_2 can reach v_1, via any b_i, i.e., (v_2, b_i, v_1) is a journey, and
- all other (temporal) connections are direct.

Definition 4.29. We denote by R_i and call it the i^{th} *theta subgraph of* $R(v_1, v_2)$ the labelled subgraph of $R(v_1, v_2)$ induced by the vertices v_1, v_2, a_i, and b_i, for some $i = 1, \dots, k$ (cf. Figure 4.13).

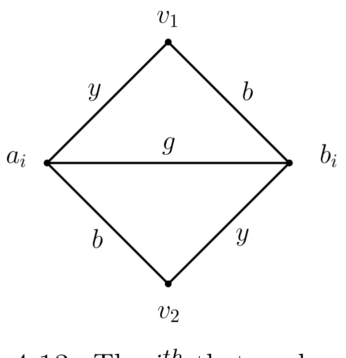


FIGURE 4.13: The i^{th} theta subgraph, R_i.

Note that the following Lemma holds:

Lemma 4.30. *Let I be an instance of $G_{n,p}$ with a uniform random labelling from the set $\{1, 2, \dots, \alpha\}$, $\alpha \geq 4$. Fix two vertices v_1, v_2 and $2k$ vertices a_i, b_i, $i = 1, \dots, k$, in I. Then, for each particular $i = 1, \dots, k$, the probability that the subgraph induced by the vertices v_1, v_2, a_i, b_i is a theta subgraph is:*

$$Pr[R_i \text{ exists in } I] \geq \left(\frac{p}{7}\right)^5$$

Proof. Each edge of R_i is realized in $G_{n,p}$ with probability p and receives the correct type of label (green, yellow, blue, or red) with probability at least $\frac{1}{7}$. Note also that the edges of different theta subgraphs R_i and R_j, $i \neq j$, are disjoint. Thus, in $G_{n,p}$, the random experiments of each of the theta subgraphs R_i appearing are *independent* from each other and each succeeds with probability at least $\left(\frac{p}{7}\right)^5$. ■

Now, consider the set of vertices $V \setminus \{v_1, v_2\}$ and partition it into two almost equal sets V_1 and V_2; note that $|V_i| \geq \lfloor \frac{n}{2} \rfloor - 2 \geq \frac{n}{3} = n'$, $i = 1, 2$. Consider a pairing of n' vertices of V_1 to n' vertices of V_2 and let the n' different pairs be the possible pairs of vertices a_i, b_i in a theta subgraph R_i. By Lemma 4.30 and since the random experiments are independent, the number of appearances of R_i is at least the number of successes in a Bernoulli distribution of n' trials, with success probability $\left(\frac{p}{7}\right)^5$ per trial. Therefore, by the Chernoff bound, we have the following Lemma:

Lemma 4.31. *Consider an instance I of a $G_{n,p}$ that has been labelled uniformly at random and fix two vertices v_1, v_2 in I. The probability that there is a temporal router $R(v_1, v_2)$ consisting of at least $k = \frac{n'}{2}\left(\frac{p}{7}\right)^5 = \frac{n}{6}\left(\frac{p}{7}\right)^5$ theta subgraphs R_i is at least $1 - e^{-\frac{1}{8}n'\left(\frac{p}{7}\right)^5} = 1 - e^{-\frac{n}{24}\left(\frac{p}{7}\right)^5}$.*

Corollary 4.32. *Note that, again by the Chernoff bound, k asymptotically almost surely* does not exceed $\frac{3}{2}n'\left(\frac{p}{7}\right)^5$, *since $Pr[k > \left(1 + \frac{1}{2}\right)E(k)] \leq e^{-\frac{n}{36}\left(\frac{p}{7}\right)^5}$.*

We now condition on the event, $\mathcal{E}_1$, that the instance I of a labelled $G_{n,p}$ has a temporal router $R(v_1, v_2)$ of at least $k = \frac{n}{6}\left(\frac{p}{7}\right)^5$ theta subgraphs. By Lemma 4.31, we know that:

$$Pr[\bar{\mathcal{E}_1}] \leq e^{-\frac{n}{24}\left(\frac{p}{7}\right)^5}.$$

Given that $R(v_1, v_2)$ exists, any vertex u that is not in $R(v_1, v_2)$ can reach any vertex u' that is also not in $R(v_1, v_2)$ *through* $R(v_1, v_2)$ via a journey, if u connects to some a_i directly with a green edge, and b_i connects to u' directly with a red edge. Then, (u, a_i, v_1, b_i, u') is a journey.

The probability of the edge $\{u, a_i\}$ being green *and* the edge $\{b_i, u'\}$ being red, for any i, is $\left(\frac{p}{7}\right)^2$ and it is independent of edge experiments inside $R(v_1, v_2)$. So, we have:

Lemma 4.33. *Condition on the event $\mathcal{E}_1$ of the existence of $R(v_1, v_2)$ in $G_{n,p}$ with at least $k = \frac{n}{6}\left(\frac{p}{7}\right)^5$ theta subgraphs. Let u, u' be any two (different) vertices of G*

that are not in $R(v_1, v_2)$. Then:

$$Pr[\textit{there exists a } (u,u')\textit{-journey through } R(v_1,v_2)] \geq 1 - \left(1 - (\frac{p}{7})^2\right)^k$$

Proof. For the vertices u, u' as described above and any one of the k possible journeys of the form (u, a_i, v_1, b_i, u'), the probability that such a journey fails (i.e., is not realized) is at most $1 - \left(\frac{p}{7}\right)^2$. Therefore, given $\mathcal{E}_1$, we have:

$$Pr[\text{there exists no } (u,u')\text{-journey through } R(v_1,v_2)] \leq \left(1 - (\frac{p}{7})^2\right)^k$$

■

Let $\mathcal{E}_2$ be the event that given k pairs of vertices a_i, b_i in a possible $R(v_1, v_2)$, each vertex pair u, u', with $u \neq u'$ and $u, u' \notin V\left(R(v_1, v_2)\right)$, satisfies the following: there is at least one pair of vertices a_i, b_i such that u connects to a_i with a green edge *and* u' connects to b_i with a red edge.

Notice that $\bar{\mathcal{E}_2}$ is the event that there is a pair of vertices u, u' that are not in $R(v_1, v_2)$ that fails to connect as described above. Since the number of possible pairs of vertices u, u' is less than n^2, we have:

$$\begin{aligned} Pr[\bar{\mathcal{E}_2}] &\leq n^2 \left(1 - (\frac{p}{7})^2\right)^k \\ &\leq n^2 e^{-k(\frac{p}{7})^2} \\ &= e^{-k(\frac{p}{7})^2 + 2\ln n} \end{aligned}$$

Now, condition on $\mathcal{E}_1$ and on $\mathcal{E}_2$ (given $\mathcal{E}_1$). Then, for each vertex $u \notin V\left(R(v_1, v_2)\right)$, keep one of its green edges (to some a_i) and one of its red edges (to the corresponding b_i), since by $\mathcal{E}_2$, those exist. Then, remove all edges of I except for the edges of $R(v_1, v_2)$ and the two edges we keep for every vertex that is not in $R(v_1, v_2)$. Notice that the resulting labelled subgraph of I is temporally connected, since:

a) $R(v_1, v_2)$ is temporally connected itself, by construction,

b) any $u \notin V\left(R(v_1, v_2)\right)$ has a journey via $R(v_1, v_2)$ to any other $u' \in V$ in the graph,

c) any a_i or b_j can reach any $u \notin V(R(v_1, v_2))$ via a journey through v_1(using first a green edge, if we start from a b_j vertex, and then using a yellow, a blue and a red edge to reach u), and

d) v_1 and v_2 can reach any $u \notin V(R(v_1, v_2))$ via a journey through some vertex b_i (using first a blue -or yellow, respectively- edge to b_i, and then a red edge to u).

The temporally connected instance I of $G_{n,p}$ *after* the removal of redundant edges as described above has a number of labelled edges (i.e., time-edges) that is at most $2n + \Theta(k)$. Since $k = \frac{n}{6}\left(\frac{p}{7}\right)^5$, I has at most $\Theta(n + np^5)$ labels after the removal of the redundant edges.

Recall that we require $p \geq 7\left(\frac{\gamma \ln n}{n}\right)^{\frac{1}{7}}$, $\gamma \geq 24$. Therefore, we get the following:

Theorem 4.34. *Consider a $G_{n,p}$, with $p \geq 7\left(\frac{\gamma \ln n}{n}\right)^{\frac{1}{7}}$, for some $\gamma \geq 24$, labelled uniformly at random. Then, any instance I of $G_{n,p}$ needs only $\Theta(n + np^5)$ time-edges to be temporally connected, with probability at least $1 - 2e^{-\frac{\gamma}{24}\ln n}$.*

Proof. Any instance I of $G_{n,p}$ becomes temporally connected by using at most $\Theta(n + np^5)$ edges (and, thus, labels) as described above, with probability at least:

$$Pr[\mathcal{E}_1] \cdot Pr[\mathcal{E}_2 | \mathcal{E}_1] \geq \left(1 - e^{-\frac{n}{24}\left(\frac{p}{7}\right)^5}\right) \cdot \left(1 - e^{-k\left(\frac{p}{7}\right)^2 + 2\ln n}\right). \tag{4.4}$$

Since $p \geq 7\left(\frac{\gamma \ln n}{n}\right)^{\frac{1}{7}}$ and $k = \frac{n}{6}\left(\frac{p}{7}\right)^5$, we have:

$$\begin{aligned} k\left(\frac{p}{7}\right)^2 &= \frac{n}{6} \cdot \left(\frac{p}{7}\right)^7 \\ &\geq \frac{n}{6} \cdot \frac{\gamma \ln n}{n} \\ &= \frac{\gamma \ln n}{6}. \end{aligned} \tag{4.5}$$

Therefore, from relations (4.4) and (4.5), we have:

$$\begin{aligned} Pr[\mathcal{E}_1] \cdot Pr[\mathcal{E}_2 | \mathcal{E}_1] &\geq \left(1 - e^{-\frac{n}{24}\left(\frac{p}{7}\right)^5}\right) \cdot \left(1 - e^{-\frac{\gamma \ln n}{6} + 2\ln n}\right) \\ &\geq \left(1 - e^{-\frac{n}{24}\left(\frac{p}{7}\right)^7}\right) \cdot \left(1 - e^{-\frac{\gamma \ln n}{6} + 2\ln n}\right) \end{aligned} \tag{4.6}$$

Again, since $p \geq 7\left(\frac{\gamma \ln n}{n}\right)^{\frac{1}{7}}$, we have that: $\left(\frac{p}{7}\right)^7 \geq \frac{\gamma \ln n}{n}$, so relation (4.6) becomes:

$$\begin{aligned}
Pr[\mathcal{E}_1] \cdot Pr[\mathcal{E}_2|\mathcal{E}_1] &\geq \left(1 - e^{-\frac{n}{24}\frac{\gamma \ln n}{n}}\right) \cdot \left(1 - e^{-\frac{\gamma \ln n}{6} + 2\ln n}\right) \\
&\geq 1 - e^{-\frac{\gamma \ln n}{6} + 2\ln n} - e^{-\frac{n}{24}\cdot\frac{\gamma \ln n}{n}} \\
&= 1 - e^{-\ln n\left(\frac{\gamma}{6} - 2\right)} - e^{-\frac{\gamma}{24}\cdot \ln n} \\
&= 1 - n^{-\left(\frac{\gamma}{6}-2\right)} - n^{-\frac{\gamma}{24}}. \qquad (4.7)
\end{aligned}$$

Now, since $\gamma \geq 24$, we have that $\frac{\gamma}{6} - 2 \geq \frac{\gamma}{24}$. Therefore, from relation (4.7), we get:

$$\begin{aligned}
Pr[\mathcal{E}_1] \cdot Pr[\mathcal{E}_2|\mathcal{E}_1] &\geq 1 - 2n^{-\frac{\gamma}{24}} \\
&= 1 - 2e^{-\frac{\gamma}{24}\ln n}.
\end{aligned}$$

■

Note that for the sparsest possible $G_{n,p}$ here, i.e., for $p = 7\left(\frac{\gamma \ln n}{n}\right)^{\frac{1}{7}}$, we need only $\Theta(n + n^{\frac{2}{7}}(\ln n)^{\frac{5}{7}}) = \Theta(n)$ edges (and, thus, labels) to satisfy TC, with probability at least $1 - 2e^{-\frac{\gamma}{24}\ln n}$, $\gamma \geq 24$.

4.5 Open problems

In a typical temporal graph design problem, a designer is given as input an underlying (di)graph G and a desired temporal graph property Π, such as maintaining all temporal reachabilities or having small maximum temporal matching. The goal then is to compute a labeling L on G such that the resulting temporal graph satisfies Π while L minimizes some cost measure. We have chosen here as our cost measure the total number of labels in L; other cost measures may include the maximum number of labels per edge (temporality [81]), the largest label used in L, or even the number of distinct labels used in L.

Alternative design problems may involve giving the designer a temporal graph that satisfies some property and asking to minimise the cost measure without violating the property; an example of such a problem is the problem of computing the removal profit in a graph that satisfies TC. Here, we have shown the latter to be APX-hard but it is still unanswered if there is a polynomial-time constant factor approximation for it.

Although in the worst case a temporal graph optimisation problem may be hard to solve, or even hard to approximate, an optimal solution may be efficiently computable in temporal graphs that occur in some particular application domain. So, designing efficient algorithms for specific temporal graph optimisation problems by imposing additional structural constraints on the input underlying graph or the labelling itself is a natural and useful alternative to approximation algorithms for efficient computation in temporal graphs. In our case, therefore, it would be interesting to examine the complexity of computing the removal profit in special classes of graphs, e.g., planar graphs or the grid? Perhaps the very complex final structure of the graph used in our APX-hardness proof for the general case of graphs is already an indication of easier solutions for the problem on those special cases.

It may also be interesting to explore the relation between optimisation problems that seem to complement each other; an example of a problem that seems to "complement" the problem of computing the removal problem is the following: Given of a non-temporally-connected temporal graph, what is the minimum number of labels that need to be *added* to its edges so that the resulting graph is temporally connected?

Chapter 5

Connectivity of Interval Temporal Graphs

In this chapter, we present work that has been published in [3] at the 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks in 2015 (ALGOSENSORS'15). An interval temporal graph is still a graph whose links change over time, however the term *interval* means that a link may exist for one or more *time intervals* and not discrete time steps. These are called *availability intervals* of the link, after which the link does not exist until, *maybe*, a further moment in time when it starts being available again. In this model, we consider continuous time and high-speed (instantaneous) information dissemination. We say that an interval temporal graph is connected during a period of time $[x, y]$, if it is connected for all time instances $t \in [x, y]$ (instantaneous connectivity). In this Chapter, we study instantaneous connectivity issues of interval temporal graphs. We provide a polynomial-time algorithm that answers if a given interval temporal graph is connected during a time period. If the graph is not connected throughout the given time period, then we also give a polynomial-time algorithm that returns large components of the graph that remain connected and remain large during $[x, y]$; the algorithm also considers the components of the graph that start as large at time $t = x$ but dis-connect into small components within the time interval $[x, y]$, and answers how long after time $t = x$ these components stay connected and large. Finally, we examine a case of interval temporal networks on tree graphs where the *lifetimes* of links and, thus, the failures in the connectivity of the network are not controlled by us; however, we can "feed" the network with extra edges that may re-connect it into a tree when a failure happens, so that its connectivity is maintained during a time period. We show that we can with high probability maintain the connectivity of the network for a long time period by making these extra edges available for re-connection using a randomised approach. Our approach also saves some cost in the design of availabilities of the edges; here, the cost is the sum, over all extra edges, of the length of their availability-to-reconnect interval.

5.1 Overview

5.1.1 Motivation and Contribution

Assuming the availability of an edge for a whole time-interval $[t_1, t_2]$ or multiple such time-intervals, and not just for discrete moments, is a clearly natural assumption since time is indeed a continuous measure. Bui-Xuan et al. [25] consider a class of dynamic networks where the changes in the topology can be predicted in advance and in which each node and each edge comes with a list of time intervals; they give algorithms for computing foremost time-respecting paths, shortest (minimum hop count) time-respecting paths and fastest (minimum time) time-respecting paths in this model. Fleischer and Tardos [46] consider a continuous-time model of dynamic graphs and prove continuous versions of known discrete-time flow algorithms for dynamic flow problems. Fleischer and Skutella [45] and Koch et al. [73] also engage in the study of flows in a continuous-time model of dynamic graphs.

In this Chapter, we restrict our attention to *continuous time* and consider systems in which only the connections between the participating entities may change, while the entities themselves remain unchanged. So we consider graphs of a fixed vertex set, each edge e of which is available over a set of time intervals $L_e = \{[t_1, t'_1], \ldots, [t_k, t'_k]\}$. Each interval indicates a period of availability of e; the unprimed times mark the start of the availability period and the primed times mark the end. This is a model that could naturally represent several systems, such as proximity networks where a link may represent that two entities have been close to each other for some extent of time, or infrastructural systems like the Internet, or even seasonal food webs where a link that is available during a time interval may represent the fact that one species is the main food source of another for a specific period of the year.

In Section 5.2, we give a polynomial-time deterministic algorithm that decides if a given interval temporal graph is connected during a given period; if the graph is not connected, the algorithm returns the maximal interval from the beginning of the given period during which the graph stays connected. In Section 5.3, we provide a polynomial-time algorithm that decides if a given interval temporal graph has *large enough* connected components during a given time period; here, the size of the components in question is determined by a parameter provided by the user as input to the algorithm. Finally, we provide a probabilistic analysis of a scenario where the *lifetime* of the intervals assigned to the edges of a network on a tree graph are not designed via a deterministic process and are unknown to us; instead, the edges may fail unexpectedly and we are required to supply the network with more available edges so that, when a break in the connectivity of the network happens, we can re-connect it. We wish not to keep all these extra edges available for re-connection at all times, that is, we wish to maintain connectivity but also pay a low cost on keeping extra edges available. Assuming that the cost of keeping additional edges available is linear to the sum of lengths of their availability intervals, we show a low cost construction.

5.1.1.1 Relation to fully-dynamic connectivity

In a fully-dynamic graph problem, a graph G over a fixed vertex set V of size n is considered, which may be updated by insertions and deletions of edges. The problem of fully-dynamic graph connectivity is to answer whether a graph remains connected under such updates, i.e., edge deletions and additions. Clearly, the latter is very similar to the problem of checking whether an interval temporal graph remains connected over time, since connectivity in interval temporal graphs is instantaneous (contrary to the temporal graphs studied in the previous chapters). Indeed, some of the proposed algorithms for solving fully-dynamic graph connectivity can be applied to the problem we consider here; we have chosen to present our algorithm for checking for connectivity of interval temporal graphs, even though it gives a worse time complexity, as an introduction to the algorithm presented in Section 5.3, which follows the same logic.

Fully-dynamic graph connectivity has been studied under various settings over the years and the related literature goes back to the 1980s if not even earlier. Holm et al. [63] consider on-line updates and queries, and give an algorithm for maintaining a spanning forest in a graph in amortized time $O(\log^2 n)$ per update, and then answering connectivity queries in time $O(\frac{\log n}{\log \log n})$. They also give efficient solutions to the fully dynamic minimum spanning forest problem, where the edges also have weights. Thorup [99] gives a near-optimal randomized and amortized algorithm for the problem of fully-dynamic graph connectivity. Henzinger and King [61] had previously showed that a spanning forest could be maintained in $O(\log^3 n)$ expected amortized time per update; then connectivity queries can be answered in $O(\frac{\log n}{\log \log n})$ time. The update time was further improved to $O(\log^2 n)$ by Henzinger and Thorup [62]. Other relevant studies include the work of Fredman and Henzinger [51], the work of Frederickson [50], and the work of King [70].

5.1.2 Preliminaries

We focus here on graphs with edges that are not always available and their availability is described by a set of time intervals, one set L_e per edge (arc) e.

Definition 5.1 (Interval temporal graph)**.** Let $G = (V, E)$ be a (di)graph. An interval temporal graph on G is an ordered triplet $G_{\mathcal{I}}(L_{\mathcal{I}}) = (V, E, L_{\mathcal{I}})$, where $L_{\mathcal{I}} = \big\{L_e = \{[t_1, t'_1], \ldots, [t_{k_e}, t'_{k_e}]\},$ for some $k_e \in \mathbb{N},\ t_i, t'_i \in \mathbb{R}^+,\ i = 1, 2, \ldots,\ k_e :$ $e \in E\big\}$ is an *assignment* of availability intervals to the edges (arcs) of G. $L_{\mathcal{I}}$ is called an *interval labelling* of G. For this Chapter and for simplicity, we call an interval labelling just “labelling”.

The availability intervals of an edge (arc) e represent the *continuous time intervals* at which e is active. When we say that an edge (arc) is active or available during the interval $[a, b]$, for some $a, b \in \mathbb{N}$, we mean that the edge exists in the graph $\forall t \in \mathbb{R}^+,\ t \in [a, b]$. For the analysis throughout this chapter, we assume the intervals $[t_1, t'_1], \ldots, [t_{k_e}, t'_{k_e}]$ to be disjoint. We can assume that, because if an edge $e \in E$ has overlapping availability intervals, then we can consider their union as an availability interval of e.

A basic assumption that we follow is that when a message or an entity passes through an available link at time t, then it can pass through a subsequent link only at some time $t' \geq t$ and only at a time at which that link is available. Notice the difference here to the model of discrete time, where we would require a subsequent link to be available only *after* time t. Also, unlike what is assumed in the discrete-time model of the previous Chapters, here we consider instant information dissemination through a path of the underlying (di)graph, if the consecutive edges (arcs) are consistently labelled. In fact, our model considers very high speed of information dissemination, resembling fibre-optic communication, but the small time needed to send a message through a link is considered negligible for the analysis. Consider, for example, the interval temporal graph of Figure 5.1, where the availability intervals of each edge are drawn next to the edge; here, information may start at time $t = 2.5$ from vertex u and arrive at time $t = 2.5$ at vertex z, since all three consecutive edges are available at time $t = 2.5$ (their availability intervals are overlapping).

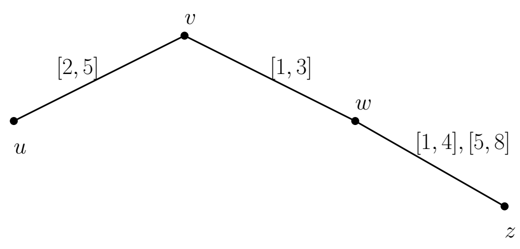


FIGURE 5.1: A model of “Fibre-optic”-like communication.

Definition 5.2 (Connectivity of interval temporal graphs)**.** An interval temporal graph $G_{\mathcal{I}}(L_{\mathcal{I}}) = (V, E, L_{\mathcal{I}})$ is connected at a given time instance t_0 if the edges that are available at time t_0, i.e., the edges that have an availability interval which includes t_0, include a spanning tree.

5.2 Connectivity of interval temporal graphs during a given time period

A fundamental issue for any given graph, dynamic or not, is to verify if the graph is connected (over time, in the dynamic case), i.e., information can travel via edges between any ordered pair of vertices in it. In this section, we consider interval temporal graphs and address the issue of their connectivity.

One can think of an interval temporal graph as a dynamic graph, where the changes in the topology of the graph happen whenever an availability interval of an edge starts or finishes, but can view it as static in between these (instantaneous) changes. Since information can travel instantaneously in interval temporal graphs, for such a graph to be connected over a time period, all the instances of the “static” graphs that are formed during that period need to be connected.

We provide below a polynomial-time procedure to determine if a given interval temporal graph is connected throughout a particular time period. Henceforth, we denote by $E(t)$ the set of edges that are available at time t, and t is *not* the finish time of the availability interval that includes t.[1]

Theorem 5.3. *There is a polynomial-time algorithm (see Algorithm 5) which, given an interval temporal graph $G_{\mathcal{I}}(L_{\mathcal{I}})$ on n vertices and numbers $x, y \in \mathbb{R}^+$, $x < y$, answers whether $G_{\mathcal{I}}(L_{\mathcal{I}})$ is connected during the time period $[x, y]$, i.e., is connected for every time instance $t \in [x, y]$. If for some $a \in [x, y]$, the maximal sub-period of $[x, y]$ during which $G_{\mathcal{I}}(L_{\mathcal{I}})$ remains connected is $[x, a] \subseteq [x, y]$, then the algorithm also returns the length of that period, namely $a - x$.*

5.2.1 Description of the algorithm

The idea behind Algorithm 5 is that $G_{\mathcal{I}}(L_{\mathcal{I}})$ is connected during a period $[x, y]$ if and only if $G_{\mathcal{I}}(L_{\mathcal{I}})$ has a spanning tree for every time instance in $[x, y]$.

Initially, Algorithm 5 finds a spanning tree of the input graph $G_{\mathcal{I}}(L_{\mathcal{I}})$ at time x. If no such tree exists, then at time x the graph is disconnected and the algorithm terminates. If a spanning tree T exists at time x, then T remains connected until one (or more) of its edges stop being available. Denote by b_1 the first moment in time at which T disconnects. T consists now of a number of connected components and, in fact, T is a forest (collection of trees). The algorithm checks whether there are edges of $G_{\mathcal{I}}(L_{\mathcal{I}})$ that are available at time b_1, which can be added to T and re-connect it. More specifically, it finds an edge that is available at time b_1 and has endpoints in different connected components of T. The algorithm adds that edge to T and checks if this addition re-connects it. If not, then it looks for yet another edge that is available at time b_1 and has endpoints in different connected components of (the current) T. This process continues until T is re-connected or we cannot find any more edges of $G_{\mathcal{I}}(L_{\mathcal{I}})$ that are available at time b_1 and have endpoints in different connected components of T. If at any step of the process there do not exist edges that can re-connect T, then the algorithm returns that the graph is disconnected. However, if it can find appropriate edges to re-connect T, then it forms another spanning tree of the graph, available from time

[1] $E(t)$ are the edges that are available at t and do not stop being available (immediately) after time t.

Algorithm 5: Connectivity of interval temporal graphs

```
Input: A temporal graph $G_{\mathcal{I}}(L_{\mathcal{I}})$ of $n$ vertices and numbers $x, y \in \mathbb{R}^+$ such that
       $x < y$
Output: Answer if $G_{\mathcal{I}}(L_{\mathcal{I}})$ remains connected during the time interval $[x, y]$
1  if $E(x)$ induces a spanning tree, $T$, of $G$ then
2      Sort the edges in $T$ according to the finish time of their availability interval;
       /* For every edge in $T$, we only consider the interval that
       includes $x$ */
3      Let $A = \{e_i,$ with interval $[a_i, b_i] : i = 1, \dots, n-1\}$ be the sorted list;
4      if $b_1 \geq y$ then /* If all edges in $T$ remain available until time $y$
       */
5          return "Graph is connected" and "Duration of survival =" $y - x$;
6      else
7          $E' := \{e \in E(T) : b_e = b_1\}$;
           /* $b_1$ is the first time instance at which $T$ becomes
           disconnected.  $E'$ is the set of edges of $T$ that stop being
           available at time $b_1$.  */
8          $T := T \setminus E'$;
           /* $T$ is now a forest, i.e., consist of a collection of trees
           */
9          Remove $E'$ from $A$;
10         Let $T_1, T_2, \dots, T_i, i \in \mathbb{N}$ be the connected components of $T$;
11         while $T$ is disconnected do
12             if $\exists j, k = 1, 2, \dots, i : \exists e = (u, v) \in E(b_1) : u \in V(T_j) \wedge v \in V(T_k)$
               then /* If there is an edge of $G$ with endpoints in
               different connected components of $T$ and is available at
               time $b_1$, then add it to $T$ */
13                 Find the $T_j, T_k$ trees of $T$ that $e$ connects;
14                 Merge $T_j, T_k$ and $e$ into a single tree;
15                 Update the number $i$ of connected components of $T$;
16                 Insert $e$ in the sorted list $A$;
17             else
18                 return "Graph is disconnected" and "Duration of survival ="
                   $b_1 - x$;
19                 Break;
20             end if
21         end while
22         Go to line 4;
23     end if
24 else
25     return "Graph is disconnected" and "Duration of survival =" 0;
26 end if
```

b_1 onwards, and the same procedure continues. The algorithm answers that the graph is connected if it forms a spanning tree, all the edges of which are available until the end of the period in question, namely until time y.

Note that Algorithm 5 not only answers whether the given graph is connected for all times $t \in [x, y]$, but also gives the duration of connectivity of the graph (survival duration) within the interval $[x, y]$.

5.2.2 Correctness of the algorithm

Clearly, by the definition of connectivity in the model of interval temporal graphs, $G_{\mathcal{I}}(L_{\mathcal{I}})$ is connected during a period $[x, y]$ if and only if $G_{\mathcal{I}}(L_{\mathcal{I}})$ has a spanning tree for every time instance in $[x, y]$. We will now show that if $G_{\mathcal{I}}(L_{\mathcal{I}})$ is connected during $[x, y]$, then Algorithm 5 answers that the graph is connected, and if $G_{\mathcal{I}}(L_{\mathcal{I}})$ is not connected during $[x, y]$, then Algorithm 5 answers that the graph is disconnected.

If $G_{\mathcal{I}}(L_{\mathcal{I}})$ is disconnected at some point $t_0 \in [x, y]$, then at time t_0, the set of available edges $E(t_0)$ will induce no spanning tree. Algorithm 5 will then look for edges to reconnect the connected components of T but will fail to do so. Therefore, it will answer that the graph is disconnected and return duration of survival equal to $t_0 - x$.

Now, suppose that the given $G_{\mathcal{I}}(L_{\mathcal{I}})$ remains connected in $[x, y]$. Let T be the spanning tree that the algorithm considers at time x. If all the edges of T remain available until time y, then the algorithm will answer that the graph is connected with duration of survival equal to $y - x$. Suppose now that there are edges in T which stop existing at some moment in $[x, y)$ and let t_0 be the first such moment. The algorithm removes from T all the edges that stop existing at time t_0. T is now disconnected but still contains all the vertices of G. Let G_{t_0} be the subgraph of G induced by $E(t_0)$.[2] Clearly, G_{t_0} is connected since $G_{\mathcal{I}}(L_{\mathcal{I}})$ is connected during $[x, y]$. Also, at time t_0, T is a subgraph of G_{t_0}. We will show by contradiction that there exist edges in G_{t_0} with endpoints in different connected components of T, which can be added to T to re-connect it. Indeed, suppose we cannot do that, i.e., we have added as many edges as possible, if any, to T and we can find no more edges with endpoints in different connected components of (the current) T. Let $T_1, T_2, \ldots, T_i$, for some $i \in \mathbb{N}$ be the connected components of T at that

[2]Recall that $E(t_0)$ is the set of all available edges of $G_{\mathcal{I}}(L_{\mathcal{I}})$ at time t_0, that do not stop being available immediately after time t_0.

point. Let w, z be two vertices of G which belong in $T_j, T_k,\ j, k \in \{1, \dots, i\},\ j \neq k$, respectively. Since G_{t_0} is connected, there exists a path p from w to z in G_{t_0}. Let w' be the last vertex in p which belongs to T_j and let w'' the vertex following w' in p. Since w' and w'' are in different connected components of T and $\{w', w''\} \in E(G_{t_0})$, there is an edge, namely $\{w', w''\}$, with endpoints in different connected components of T, which we can add to T. This contradicts our assumption. Therefore, we can keep adding edges, as the algorithm indicates, until we re-connect T.

The above process goes on until time $t = y$, or until we have a spanning tree that remains intact until time $t = y$. Then, Algorithm 5 answers that the graph is connected and returns duration of survival equal to $y - x$.

5.2.3 Running time

The running time of Algorithm 5 depends on the number of times that the spanning tree changes during $[x, y]$. The spanning tree can only change when one or more edges stop being available, so the above number is in general upper bounded by the total number of intervals assigned to the edges of the graph:

$$M = \sum_{e \in E} |L_e|.$$

Initially, to find $E(x)$ we need to look at every edge $e \in E$ and decide if x is between the start and finish time of one of e's availability intervals. Performing a binary search on the ordered set of start times *and* the ordered set of finish times of e's availability intervals, we can decide if $e \in E(x)$ in time $O(\log |L_e|)$. So, to compute $E(x)$ and check if it induces a spanning tree, we need time:

$$M' = O(\sum_{e \in E} \log |L_e|).$$

Next, time $O(n \log n)$ is required to sort the edges in T, where n is the number of vertices in the graph. Then, for every time T changes, we need time M' to find the new set of available edges at the time. We need time $O(n)$ to find the connected components that can be re-connected by the addition of an available edge at the time and update T. Since we add at most $O(n)$ edges to re-connect T, the addition of all edges and the updates of T take a total of $O(n^2)$ time. Also, time $O(n)$ is required to insert the added edges in the sorted list A. Therefore, the running time of Algorithm 5 is $O(M' + n \log n + M \cdot (M' + n^2)) = O(M \cdot (M' + n^2))$.

The problem that was presented in this section could be solved faster using an existing algorithm for fully dynamic connectivity [99]; we have chosen to present our solution here as an introduction to the (similar) methodology used to solve the problem of Section 5.3.

5.3 Large connected components during a given time period

In this section, we examine if, given an interval temporal graph $G_{\mathcal{I}}(L_{\mathcal{I}})$ of n vertices, numbers $x, y \in \mathbb{R}^+$ and a parameter $0 \leq \varepsilon \leq 1$, we can find one or more *large enough* subsets of the vertices of G which remain connected within the same subset and remain large within the time interval $[x, y]$. The matter of how large we want these components to be can be handled by adjusting the parameter ε, which gives us a lower bound of $\varepsilon \cdot n$ on the size of the components we are looking for. In this section, we provide an algorithm that efficiently solves the above problem. The problem discussed in Section 5.2 is clearly a special case of the problem we consider here, namely the case where the parameter ε is equal to 1. However, we feel that the previous Section provides a smoother introduction to to the presented technique, hence, we include it as a separate case. Henceforth, a "large enough" connected component will be a component of size at least $\varepsilon \cdot n$.

Notice that any connected component C of $G_{\mathcal{I}}(L_{\mathcal{I}})$, at time $t = x$, that is not large enough can be omitted by any algorithm that solves the above problem. Even if the vertices of C connect with more vertices in $G_{\mathcal{I}}(L_{\mathcal{I}})$ at a later moment in time within $[x, y]$, resulting in a large enough connected component C' of $G_{\mathcal{I}}(L_{\mathcal{I}})$ at that time, C' is not a component that was connected throughout $[x, y]$.

Theorem 5.4. *There is a polynomial-time algorithm which, given an interval temporal graph $G_{\mathcal{I}}(L_{\mathcal{I}})$ on n vertices and numbers $x, y \in \mathbb{R}^+$, $x < y$, returns all subgraphs of G of size $\varepsilon \cdot n$, $0 \leq \varepsilon \leq 1$, that remain connected and large (i.e., is always of size at least $\varepsilon \cdot n$) during the time period $[x, y]$. If $[x, a] \subseteq [x, y]$, $a \in [x, y]$, is the maximal sub-period of $[x, y]$ during which such a component remains connected, then the algorithm also returns the length of that period, $a - x$.*

```
Algorithm 6: Large connected components in interval temporal graphs
Input: A temporal graph $G_{\mathcal{I}}(L_{\mathcal{I}})$ of $n$ vertices, numbers $x, y \in \mathbb{R}^+$ such that
       $x < y$ and parameter $\varepsilon :\ 0 \leq \varepsilon \leq 1$
Output: All components of $G_{\mathcal{I}}(L_{\mathcal{I}})$ of size at least $\varepsilon \cdot n$ that remain connected
        during the time interval $[x, y]$
1  Find the set $E(x)$ of available edges at time $x$, distinguish the connected
   components and delete those of size smaller than $\varepsilon \cdot n$
2  for each of the remaining connected components do
3      Find a spanning tree, $T$
4      $n' = |V(T)|$
5      Sort the edges in $T$ according to the finish time of their availability interval
       Let $A = \{e_i,$ with interval $[a_i, b_i] :\ i = 1, \ldots, n' - 1\}$ be the sorted list
       // For every edge in $T$, we only consider the interval that
       includes $x$.
6      OutputComponents(T,A,y)
7  Function OutputComponents(a tree $T$ of $n'$ vertices, the sorted list
   $A = \{e_i,$ with interval $[a_i, b_i] :\ i = 1, \ldots, n' - 1\}$ of edges in $T$, parameter
   $y \in \mathbb{R}^+$)
8      if $b_1 \geq y$ then
9          Output V(T) and "Duration of survival of component = " $y - x$
10     else
11         $E' := \{e \in E(T) : b_e = b_1\}$ // $b_1$ is the first time instance at
           which $T$ becomes disconnected. $E'$ is the set of edges of $T$
           that stop being available at time $b_1$.
12         $T := T \setminus E'$
13         Remove $E'$ from $A$
14         Let $T_1, T_2, \ldots, T_i, i \in \mathbb{N}$ be the connected components of $T$
15         while $T$ is disconnected and $|V(T)| = n'$ do
16             if $\exists j, k = 1, 2, \ldots, i : \exists e = (u, v) \in E(b_1) : u \in V(T_j) \wedge v \in V(T_k)$
               then // If there is an edge of $G$ with endpoints in
               different connected components of $T$ and is available at
               time $b_1$, then add it to $T$
17                 Find the $T_j, T_k$ trees of $T$ that $e$ connects
18                 Merge $T_j, T_k$ and $e$ into a single tree
19                 Update the number $i$ of connected components of $T$
20                 Insert $e$ in the sorted list $A$
21             else
22                 for each connected component $C$ of $T$ with size smaller than $\varepsilon \cdot n$
                   do
23                     $T = T \setminus C$
24         for each connected component, $C$, of $T$ do
25             $T := C$; $n' = |V(C)|$; $A = A \setminus$ edges that are not in $C$
26             OutputComponents(T,A,y)
```

5.3.1 Description of the algorithm

Algorithm 6 receives as input an interval temporal graph of n vertices and an interval $[x, y]$ during which we want to check whether one or more large components of the graph remain connected. The algorithm also takes a non-negative parameter ε no larger than 1. This parameter defines how large we want our components to be; more specifically, the algorithm will only look for components of size (number of vertices) at least $\varepsilon \cdot n$. The algorithm returns all those subsets of the vertices of the initial graph, if any, that remain connected (and large) during $[x, y]$. Furthermore, it returns the duration of connectivity (*survival duration*) of any large enough component that was connected at time $t = x$ but disconnects at some point in $[x, y]$.

To do so, the algorithm initially checks which connected components, if any, are large enough at time x, and ignores all the rest. Then, the algorithm treats each and every one of these large components similarly, but separately. Namely, for each one of them the algorithm finds a spanning tree T and sorts all its edges according to the finish time of their availability interval, considering only the interval that includes time x. If the same tree remains connected during $[x, y]$, then the algorithm returns the respective component. Otherwise, if the tree disconnects at a moment t_0 in time, the algorithm employs a similar process to the one used in Algorithm 5, i.e., tries to reconnect the remainder of the tree via edges that are available at t_0. If T cannot be re-connected, then the algorithm checks the sizes of its connected components; it ignores those that are not large enough, while "processing" the rest similarly and separately as before. For each component that is ignored in the process, the algorithm returns the duration of its survival, meaning how long its vertices stayed connected since time x. The algorithm stops when there are no more components that are large enough or when the last component stays connected until time y.

5.3.2 Correctness of the algorithm

A component (subgraph) of $G_{\mathcal{I}}(L_{\mathcal{I}})$ is connected during a period $[x, y]$ if and only if there are paths between all pairs of its vertices for all $t \in [x, y]$, i.e., if it induces a spanning tree for every time instance in $[x, y]$. We will now show that if there is a large enough component of $G_{\mathcal{I}}(L_{\mathcal{I}})$ that is connected during $[x, y]$

then Algorithm 6 will find it, and if there is no such component of $G_{\mathcal{I}}(L_{\mathcal{I}})$ then Algorithm 6 will answer that. Clearly, if at time x a connected component of $G_{\mathcal{I}}(L_{\mathcal{I}})$ is not large enough, then we can ignore it; even if at a later moment in time, its vertices connect with more vertices forming a large enough component, that newly formed component is not one that survived throughout $[x, y]$.

Suppose that no large enough connected component of the given $G_{\mathcal{I}}(L_{\mathcal{I}})$ survives through $[x, y]$. Then, either there is no large enough component connected at time x, or there are large enough connected components at time x that get disconnected at some later point $t_0 \in (x, y]$, and break into small connected components. In the first case, Algorithm 6 will delete all the components induced by $E(x)$ and terminate. In the second case, it only processes the large enough components, separately; again, we separate the components because, even if at a later moment in time their vertices connect with other vertices forming a large enough component, that newly formed component is not one that survived throughout $[x, y]$. Since no component survives through $[x, y]$, there is a breaking point for each component, at which there is no tree connecting its vertices. Algorithm 6 realises that (see line 21), removes all small sub-components and *continues processing the remaining ones separately*. Clearly, for each of these sub-components there will be a moment where they disconnect without possibility of re-connection and with all their components being not large enough; when that happens, and after the execution of the for-loop in line 22, the component becomes empty and the calculation moves on to the next component, if any, until all components become empty and the algorithm terminates.

Now, suppose that the given $G_{\mathcal{I}}(L_{\mathcal{I}})$ has large enough components that remain connected in $[x, y]$. Again, we can view each component separately. Let T be the spanning tree of the component C that the algorithm considers at time x. If all the edges of T remain available until time y, then the algorithm will answer that the component remains connected throughout $[x, y]$. If, however, there are edges in T that stop existing at some moment in $[x, y]$, then the algorithm considers the first such moment, t_0. It removes from T all the edges that stop existing at time t_0. T is now disconnected but still contains all the vertices of the component C. If the subgraph of $G_{\mathcal{I}}(L_{\mathcal{I}})$ induced by the vertices of T and the edges between them available at time t_0 is connected, then the algorithm finds edges to reconnect T (analysis is similar to the analysis of Algorithm 5). If, however, the subgraph is not connected, then the algorithm connects as many components of the subgraph as can be connected (argument similar to the one used in Algorithm 5); it then ignores the small components and proceeds to process the large enough components. Since $G_{\mathcal{I}}(L_{\mathcal{I}})$ has large enough components that remain connected in $[x, y]$, eventually the algorithm finds the component(s) that remain connected throughout $[x, y]$ and outputs them (see line 9), before moving on to any remaining components, if any, that need to be checked (e.g., remaining components from the for-loop in line 24 or from the initial for-loop in line 2).

5.3.3 Running time

The running time of Algorithm 6 depends on the number of times that the spanning trees of the connected components change during $[x, y]$. The spanning trees can only change when one or more of their edges stop being available. The edges on different spanning trees are not overlapping, so the above number is in general upper bounded by the total number of intervals assigned to the edges of the graph:

$$M = \sum_{e \in E} |L_e|.$$

Initially, to find $E(x)$ we need to look at every edge $e \in E$ and decide if x is between the start and finish time of one of e's availability intervals. Performing a binary search on the ordered set of start times *and* the ordered set of finish times of e's availability intervals, we can decide if $e \in E(x)$ in time $O(\log |L_e|)$. So, to compute $E(x)$, distinguish the connected components and remove those that are not large enough, we need time:

$$M' = O(\sum_{e \in E} \log |L_e|).$$

After that, the algorithm proceeds for each component separately. It is easy to see that the running time of the algorithm for each separate component is the same as the running time of Algorithm 5. Since there are at most $\frac{1}{\varepsilon}$ connected components of size at least $\varepsilon \cdot n$ in $G_{\mathcal{I}}(L_{\mathcal{I}})$ during $[x, y]$, the running time of Algorithm 6 is $O\left((M \cdot (M' + n^2))\right)$.

5.4 Low cost maintenance of a tree structure

In this section, we consider an interval temporal network on an underlying complete graph of n nodes, i.e., all $\binom{n}{2}$ links between nodes of the network *may* exist.

The connectivity of the network needs to be maintained at all moments in time via a tree structure, i.e., a spanning tree of the clique. Each node of the tree performs an individual application determined by the operator of the structure and each link (edge) is active, i.e., "alive", during a time-interval also decided by the operator, after which the link fails. We have the liberty to provide the operator with extra edges to re-connect a spanning tree when a link fails; note here that after a new edge is added to the tree structure, the operator then assigns to it a *new* "lifetime" interval, which is determined by the application. The extra edges that we can provide at any point come from the edges of the complete graph on the set of n nodes, i.e., a total of $\binom{n}{2}$ edges. We need to assign to every edge of the clique one or more availability intervals, with I_e being the union of intervals for each edge e, so that when the tree structure becomes dis-connected, there is an appropriate such edge available to re-connect it. We allow edges to have more than one availability intervals, and we also allow edges to have no availability interval. We call those edges *reserved* edges and the set that consist of all those edges (with their availability intervals, if any) *reservoir*, denoted by R.

Notice here the distinction between the *availability* of an edge and the *lifetime* of an edge: availability refers to the interval that we assign to a reserved edge with the purpose to re-connect the tree when it breaks, and lifetime refers to the interval that the operator assigns to an edge after it is inserted in the tree structure and is the time interval after which the respective link in the tree structure will fail.

Definition 5.5 (Cost of the reservoir)**.** The cost of the reservoir is defined as the sum, over all edges in the reservoir, of the length of the edges' availability interval(s):

$$c = \sum_{e \in R} |I_e|.$$

Let T be the tree structure that is handled by the operator. We consider the time period between 0 and n and we assume that the breaks/failures in the connectivity of T happen once inside every consecutive time interval of length $\alpha \log n$, for some $\alpha > 1.2$ (*Low-frequency-of-link-breaks assumption*). We are not able to predict when exactly the failures happen, nor are we able to foresee which link will fail next. Our goal is to design the availabilities of the edges in R so that whenever a failure happens, there is an available edge to re-connect T. Once we design R, it then remains fixed until the end of the time interval $[0, n]$.

The trivial design of the availabilities of the reserved edges would be to make them all available throughout the considered time period $[0, n]$. However, this yields cost:

$$c = \sum_{i=1}^{\binom{n}{2}} n \in O(n^3).$$

We will give a design of the availabilities of the reserved edges with lower cost than the trivial, so that the network connectivity is maintained asymptotically almost surely.

Theorem 5.6. *If failures of the edges happen once in every consecutive* $\alpha \log n$, $\alpha > 1.2$ *time-intervals, then there exists a reservoir of cost* $O(n^2 \log n)$ *that keeps a spanning tree available during* $[0, n]$ *asymptotically almost surely.*

Proof. Partition the time interval $[0, n]$ into consecutive equisized sub-intervals $b_1, b_2, \ldots, b_{\frac{n}{\alpha \log n}}$ of length $\alpha \log n$, called *boxes.*[3]

Select $m' = \frac{\alpha n \log n}{2}$ edges uniformly at random, out of the $\binom{n}{2}$ reserved edges in R, to be available in b_1.
Independently of the selection of the edges in b_1, select $m' = \frac{\alpha n \log n}{2}$ edges uniformly at random, out of the $\binom{n}{2}$ reserved edges in R, to be available in b_2.
Continue in the same fashion, until there are m' edges in every box; notice that some edges may have been chosen to be available in several boxes.

The total cost of R is:

$$c = \left(\frac{\alpha n \log n}{2} \cdot \frac{n}{\alpha \log n} \right) \cdot \alpha \log n = \frac{\alpha}{2} n^2 \log n.$$

Every time an edge fails in the tree, we look amongst the edges of R that are available in the box that includes the moment of failure. If we find an edge e that can re-connect T, we add it to the structure; notice that e is not removed from R, so if in a subsequent moment of failure in T, e happens to be available during the respective box, it can be examined again for potential re-connection.

[3]Here, the last box is not necessarily of size exactly $\alpha \log n$ but this does not affect the analysis.

We will now show that when a failure happens in T, we can asymptotically almost surely find an edge in R which is available at that particular moment in time to successfully re-connect the tree.

Consider the specific box b_i that includes the time moment at which the failure in T happens. Clearly, the number of edges in b_i that can re-connect T depends on where the failure happens, i.e., what the sizes[4] of the two connected components of T after the failure are. If n_1 and n_2 are the sizes of the connected components of T after a failure, then the probability that a particular edge $e \in b_i$ can re-connect T after being added to the structure is (since e was randomly chosen out of the edges of the clique):

$$\begin{aligned} Pr[e \in b_i \text{ re-connects } T] &= \frac{\text{all edges that can re-connect}^5}{\text{all edges that are available for use}^6} \\ &= \frac{n_1 \cdot n_2}{\binom{n_1+n_2}{2}} \\ &= \frac{n_1 \cdot n_2}{\frac{(n_1+n_2)\cdot(n_1+n_2-1)}{2}} \\ &\geq \frac{2n_1n_2}{(n_1+n_2)^2}. \end{aligned}$$

The probability that no edge of b_i reconnects T after a failure is:

$$Pr[\text{no } e \in b_i \text{ re-connects } T] \leq \left(1 - \frac{2n_1n_2}{(n_1+n_2)^2}\right)^{m'}.$$

So, the probability that there is an edge in b_i that re-connects T is:

$$Pr[b_i \text{ re-connects } T] \geq 1 - \left(1 - \frac{2n_1n_2}{(n_1+n_2)^2}\right)^{m'}.$$

In the best case, T dis-connects into two equisized connected components, and in the worst case it dis-connects into a component of size $n-1$ and a single vertex. So, in the worst case, we can reconnect T after a failure with probability:

$$Pr[b_i \text{ re-connects } T] \;\geq\; 1 - \left(1 - \frac{2(n-1)}{n^2}\right)^{m'}$$

[4]The size of a component is the number of its vertices.
[5]These are all the edges between the two connected components.
[6]These are all the edges in R.

$$\begin{aligned}
&= 1 - \left(1 - \frac{2(n-1)}{n^2}\right)^{\frac{\alpha n \log n}{2}} \\
&\geq 1 - e^{-\frac{2(n-1)}{n} \cdot \frac{\alpha \log n}{2}} \\
&= 1 - n^{-\frac{\alpha(n-1)}{n}} \\
&\geq 1 - n^{-0.9\alpha} \text{ , for } n \geq 10.
\end{aligned}$$

So, the probability that b_i fails to reconnect T is:

$$Pr[b_i \text{ fails to re-connect } T] \leq n^{-0.9\alpha}.$$

Let $\mathcal{E}$ be the event that "for every box b_i, $i = 1, 2, \ldots, \frac{n}{\alpha \log n}$, b_i re-connects T (when the failure happens at a moment during b_i)".

Notice that $\bar{\mathcal{E}}$ is the event that "there is a box b_i s.t. b_i fails to re-connect T".

It holds that:

$$\begin{aligned}
Pr[\bar{\mathcal{E}}] &\leq \#\text{boxes} \cdot Pr[b_i \text{ fails to re-connect } T] \\
&\leq \frac{n}{\alpha \log n} \cdot n^{-0.9\alpha} \\
&= \frac{1}{\alpha n^{0.9\alpha - 1} \log n}.
\end{aligned}$$

Therefore, the probability that the reservoir availabilities we designed can re-connect T within the time period $[0, n]$ is:

$$Pr[\mathcal{E}] \geq 1 - \frac{1}{\alpha n^{0.9\alpha - 1} \log n} \xrightarrow[\alpha > 1.2]{n \to +\infty} 1$$

So, we can almost surely re-connect T during $[0, n]$ by employing the above random assignment of the edge availability intervals, having a total cost $c \in O(n^2 \log n)$.

■

5.5 Open problems

In this Chapter, we investigated issues that have to do with the connectivity of interval temporal graphs. There is still potential for more efficiently running algorithms for checking and outputting large connected components of an interval temporal graph over time; one could, for example, try to exploit algorithms for fully dynamic graph connectivity problems to try and achieve better performance.

With respect to the issue of maintaining an underlying tree connected over time, while at the same time keeping the cost at low levels, it is yet unanswered whether there is a more efficient way than one that costs $O(n^2 \log n)$. One could try to explore a lower bound on the cost of any such design, or even explore other patterns of link failures. We have assumed here that links in the tree structure fail once in every time interval of length $\alpha \log n$, but this is only one special case. It may be of interest to explore other application domains or other patterns that link failures may follow. For example, an application may cause links to receive a lifetime (inside the tree structure) given by the Exponential Distribution, after which the links fail. What would be an efficient reservoir design in such cases?

Chapter 6

Conclusions

6.1 Overview

This thesis has presented a variety of solutions to several problems that can all be categorised in the area of Theoretical Computer Science, and more specifically in the field of Algorithmic Graph Theory. The problems themselves have been concerned with an array of questions on Temporal Graphs (Temporal Networks) mostly in centralized settings. The models used draw their inspiration from several network-based environments, both deterministic (absolute knowledge of the setting) and random (uncertain behaviour of the network).

The results presented in the preceding Chapters are the result of published work carried out by the author, their supervisors as well as other collaborators from different institutions, with the exception of Chapter 3 which is under review at the time of the write-up of this Thesis. Below is a more detailed look at the conclusions that can be obtained from the main results presented in this thesis.

6.1.1 The Temporal Diameter of Random Temporal Graphs

Chapter 2 introduces and studies the model of random ephemeral temporal graphs. It is assumed that a particular range of time is split into equisized periods; the edges of the graphs we study become available at a number of discrete time steps during every period, and each such moment of availability of each edge is selected independently and uniformly at random. We have shown that a class of temporal graphs on instances of the Erdös-Renyi model of random graphs is classified as *fast*, and we have also provided examples of *slow* graphs. We have shown that there exist graphs, each edge of which needs to receive a large, unbounded by any constant, number of availabilities per period, for the resulting temporal graph to be fast. However, we have proven a general non-constant upper bound on the number, ρ^*, of availabilities per edge and per period needed for a graph to be fast.

An avenue forward on the study of the temporal diameter of random temporal graphs would be to *fully characterise* fast temporal graphs, in which information dissemination is expected to be very quick. Establishing a tighter upper bound on the critical availability, ρ^*, is also a problem of further research. Another issue of interest to us is to study models of random temporal networks, where the random selection of availability labels for the edges of the underlying graph follows distributions, other than the uniform.

6.1.2 Temporal Network Flows

Chapter 3 defines and studied flows in temporal networks for the first time. We have proven that in our model of temporal graphs, the problem of finding the maximum flow that can move from a source vertex to a sink vertex up to a given point in time is polynomially solvable. In the context of the study of temporal network flows, we have also considered the model of mixed temporal networks, in which there are edges with random availabilities and edges with specific, predefined availabilities. For this model, we have shown that it is #**P**-hard to compute the expected maximum truncated temporal flow. A relevant open problem is the existence of an FPTAS for the expected maximum flow value in mixed temporal networks. The study of flows in periodic temporal networks, where edges become available every x number for steps, for some x (possibly different for every edge) could also be a promising path following our research on temporal flows.

6.1.3 Small cost design of Temporal Graphs

After our work on the temporal diameter, as well as the study of temporal flows in the discrete time model of temporal graphs, it was a natural progression for us to move on to studying how to *design* efficient such temporal graphs. Our chosen measure of efficiency here is temporal connectivity. More specifically, we have studied the complexity of testing and designing nearly cost-optimal graphs that are temporally connected. Our results on the topic are presented in Chapter 4. We give an asymptotically optimal procedure to design a temporally connected graph on any given static undirected graph. We also partially characterise minimal temporal graphs, which are temporally connected but the removal of *any* edge availability would result in the graph losing its temporal connectivity. We also focus on the computation of the removal profit of temporally connected graphs, that is, the maximum number of labels that can be removed from the given temporal graph without it losing temporal connectivity.

As stated already in Chapter 4, we have not yet provided a polynomial-time constant factor approximation algorithm for the computation of the removal profit in a given temporally connected temporal graph. Therefore, investigating this further would be a natural progression of our work and would be something that would have been included in this thesis had time allowed. Further research could also investigate the complexity of computing the removal profit in special classes of graphs, e.g., planar graphs or the grid. Extensions of this research also include the study of the interval temporal networks model, where edges can be available for continuous intervals of time, as well as a more in-depth study of models of random temporal networks.

We believe that the issue of *designing* efficient temporal graphs is a promising research area, in general. There are several other parameters to consider when designing temporal graphs. An example would be the *robustness* of the design; can we assign availabilities to the edges of the underlying graph to satisfy some property $\mathcal{P}$, so that even after failures in some edges, the graph still satisfies property $\mathcal{P}$?

6.1.4 Interval Temporal Graphs

A natural research avenue for us, after working on discrete time temporal graphs, was to move on to studying other settings and to see what useful conclusions can be drawn for them. This is why we started looking at the model of interval temporal graphs. The results obtained for this model are presented in Chapter 5 and consider problems related to the connectivity of interval temporal graphs. We study the verification and maintenance of the connectivity in interval temporal networks. We pose an open question to maintain the spanning tree in the interval

$[0, n]$ with cost less than $O(n^2 \log n)$. An extension of this research includes the construction of a reservoir to maintain a spanning tree of the clique network, in cases where the failures in the links of the tree happen randomly, e.g., if each link receives a lifetime given by the Exponential Distribution.

6.2 Final Remarks

Over the course of this project, we have felt that we have contributed to the study of a new class of dynamic graphs explored within the field of Theoretical Computer Science. We have also felt that, via this contribution, we have gained a greater understanding of surrounding topics such as Dynamic Network Flows. It is our hope that the work presented here and the papers that it draws from can also spark a greater interest in the community for Temporal Graphs, as well as for the analysis of *Random* Temporal Graphs. There has already been a great amount of work done on the subject of dynamic graphs in several forms and models, but we firmly believe that many more interesting questions can be posed for the model of Temporal Graphs, especially when looking at cases where the availability of links follows some Probability Distribution.

Appendix A

Appendix - Basic Notions

This Chapter is meant for readers who are perhaps unfamiliar with the fundamental concepts of Computer Science that are closely related to the topic of this thesis. The author describes basic concepts of algorithms and graph theory that may help the reader better understand the work presented in the main chapters.

A.1 Algorithms

A.1.1 Overview

Algorithms are procedures or formulas used to solve a computation problem; they consist of self-contained step-by-step operations to be performed. More specifically, algorithms are well defined steps that receive an input, which may be a single value or a set of values, and in the end return an output, which may be either a single value or a set of values.

A.1.2 Analysis

The main purpose of constructing an algorithm to solve a problem is so that we obtain the solution *effectively*. Part of the analysis of an algorithm is measuring the algorithm's efficiency in some way. When measuring the efficiency of an algorithm, we usually refer to its speed, i.e., how fast the algorithm solves the problem.

However, one could just as easily measure the memory or energy usage as a metric of efficiency for analytical purposes.

For this work, we are interested in the speed of our algorithms, i.e., their running time. The computer on which the algorithm runs may have an effect on the actual time it takes the algorithm to finish, due to possibly different hardware; so to be independent of that, we measure the running time of our algorithms in the number of steps executed.

Definition A.1 (Steps of an algorithm)**.** Each line of code in an algorithm is considered to be a step and to take a constant time to run.

Definition A.2 (Running Time)**.** The time it takes an algorithm to complete all its steps and finish a task.

In order to better understand the performance of an algorithm for any size of input, we express its running time as a function of n, where n is the size of the input. Here, it is important to understand that when n is very large, then the lower order terms of the running time function are considered insignificant. So when analysing the running time of an algorithm and computing the respective function, we tend to only consider non-constant factors and in particular the order of growth of the function.

Definition A.3 (Order of growth)**.** The order of growth of a function is a way of specifying how the function scales as the input grows.

So, instead of specifying the exact relation between the input and the running time of an algorithm, we specify a *limit* for the algorithm's running time; for example, if the running time is $5n^2+3$, where n is the input size, we say that the running time scales as n^2. In Computer Science, we describe the limiting behaviour of functions through asymptotic notation. Informal definitions of this form of notation are shown in Table A.1.

For a good introductory book on Algorithms, the reader is referred to [33].

Table A.1: The meaning of asymptotic notations.

Notation	*Description*
$f(n) = \mathcal{O}(g(n))$	f is asymptotically bounded from above by g.
$f(n) = o(g(n))$	f is asymptotically dominated by g.
$f(n) \sim g(n)$	f is asymptotically equal to g.
$f(n) = \Omega(g(n))$	f is asymptotically bounded from below by g.
$f(n) = \Theta(g(n))$	f is asymptotically bounded from above and below by g.
$f(n) = \omega(g(n))$	f asymptotically dominates g.

A.1.3 Complexity classes

Problems with a yes-or-no answer are called *decision problems*, whereas problems that require finding the *best* solution from all feasible solutions are called *optimization problems*. We know that each optimization problem has an equivalent decision problem. For example, for the problem of finding the shortest route between two cities:

- the optimization/search version can be described as: Given the map, find the shortest route from city A to city B.
- the decision version can be described as: Given the map and a non-negative number k, is there a route from city A to city B with length at most k?

Problems for which we know efficient algorithms that solve them belong in a complexity *class*, i.e., group of problems, called **P** (problems solvable in polynomial-time). However, there are still problems, for which we have yet to know if they have efficient solutions; we call these problems **NP**-complete. Informally, **NP** is the class of all decision problems for which the instances where the answer is "yes" have efficiently verifiable proofs. More precisely, these proofs have to be verifiable by deterministic computations that can be performed in polynomial time. **NP**-complete problems have the interesting property that if an efficient solution is found for one of them, then it means that efficient solutions exist for all of them. Until we can decide if **NP**-complete problems admit (exact) efficient solutions, we develop what we call *approximation algorithms* that take us close to an efficient solution. When referring to **NP**-hard problems, we mean problems that are "at least as hard as the hardest problems in **NP**". More precisely, a decision problem H is **NP**-hard when for every problem L in **NP**, there is a polynomial-time reduction from L to H.

In computational complexity theory, the complexity class **#P** (pronounced "sharp P") is the set of the counting problems associated with the decision problems in the set NP. More formally, **#P** is the class of function problems of the form "compute $f(x)$", where f is the number of accepting paths of a non-deterministic Turing machine running in polynomial time. Unlike most well-known complexity classes, it is not a class of decision problems but a class of function problems. So, if an **NP** problem is of the form "Are there any solutions that satisfy certain constraints?", the corresponding **#P** problem asks "how many" rather than "are there any". Clearly, a **#P** problem must be at least as hard as the corresponding **NP** problem. If it's easy to count answers, then it must be easy to tell whether there are any answers; just count them and see whether the count is greater than zero. We say that a problem is **#P**-complete if and only if it is in **#P**, and every problem in **#P** can be reduced to it by a polynomial-time counting reduction. Equivalently, a problem is **#P**-complete if and only if it is in **#P**, and for any non-deterministic Turing machine, the problem of computing the number of its accepting paths can be reduced to this problem [101]. Similarly to **NP**-hard problems, **#P**-hard problems are counting problems that are "at least as hard as the hardest problems in **#P**". More formally, a counting problem H is **#P**-hard when for every problem L in **#P**, there is a polynomial-time counting reduction from L to H.

For a good book on computational complexity and the various complexity classes, the reader is referred to [92].

A.2 Graphs and networks

A.2.1 Overview

As the title reveals, the topic of this thesis is graphs (and networks). It is generally accepted to describe a network topology using a graph, the vertices and edges of which represent the communicating entities and the communication opportunities between them respectively. In this thesis, we alternate between the terms *graph* and *network* to describe the same notion:

Definition A.4 (Undirected Graph)**.** An undirected graph is an ordered pair $G = (V, E)$ comprising a set V of vertices, otherwise called nodes, and a set E of edges, otherwise called links, which are 2-element subsets of V.

We tend to just use the term *graph* to describe an undirected graph. If the graph is *directed*, then the elements of the set E are *ordered pairs* of elements of V; this gives a direction to the respective edge from the first element of the pair to the second element of the pair. A directed graph is also called a *digraph*. Another type of (di)graph, usually called a *multigraph*, allows for E to be a multiset of unordered or ordered pairs of (not necessarily distinct) elements of V.

A.2.2 Special Classes of Graphs

Depending on the properties of a graph, e.g., how the vertices are connected with each other, some classes of graphs get their own name. In this thesis, we often refer to graphs called cliques (or complete graphs) or graphs called trees.

Definition A.5 (Complete Graph or Clique)**.** A complete graph or clique is a graph in which all pairs of vertices are connected with edges.

Definition A.6 (Tree)**.** A tree is a graph that is connected, i.e., there is a continuous line (path) from every vertex to every other vertex, and contains no cycles, i.e., no closed paths.

A.2.3 Random Graphs

The term "random graph" is the general term used to refer to probability distributions over graphs. Random graphs may be described by a probability distribution, or by a random process that generates them. The theory of random graphs lies at the intersection of graph theory and *probability theory* (see Section A.4). For the purpose of this thesis, when referring to models of random graphs we focus our attention on a particular model, called *Erdös-Rényi* model $G_{n,p}$.

Definition A.7 (Erdös-Rényi graphs)**.** In the Erdös-Rényi $G_{n,p}$ model of random graphs, a graph of n vertices is constructed by connecting vertices at random as follows: each edge exists (is included) in the graph with probability p, independent from every other edge.

Models of random graphs such as the above can be used to prove the existence of graphs satisfying various properties, or to provide a careful definition of what it means for a property to hold for almost all graphs.

The reader is referred to [24, 60] for more details on Graph Theory and Random Graphs.

A.3 Centralised and Distributed Computing

Part of this work is rooted in the area of Distributed Computing, meaning that the model(s) we consider can be seen as what we call Distributed Systems.

Definition A.8 (Distributed System)**.** A non-centralised system of participating entities, e.g., networked computers with processors running in parallel, each having its own memory. The main characteristic of a distributed system is the absence of a controlling/central entity.

Distributed Computing tackles a problem or task by allowing communication between the participating entities, e.g., computers; the entities coordinate their actions by passing messages to each other so that they can eventually achieve a common goal, i.e., solve the problem. That can often prove useful, since the task may be completed much faster if split into sub-parts that can be carried out individually by each entity in the system. For more information on Distributed Computing, the reader is reffered to [14].

On the other hand, a large part of this thesis lies on the area of Centralised Computing, in the sense that it considers Centralised Structures/Networks.

Definition A.9 (Centralised System)**.** A type of system, e.g., network, where all users connect to a central server, which is the acting agent for all communications. This server would store both the communications and the user account information.

A centralised computation in our case translates in us knowing and having control over the network/structure in its entirety, while trying to perform some operation, e.g., solve a problem or optimize some parameter.

A.4 Probability Theory and Probabilistic Analysis

Probability theory is the branch of mathematics that studies the analysis of random phenomena. The central objects of probability theory are random variables, stochastic processes, and events. A random variable is a variable whose value is subject to variations due to randomness, more specifically it can take a value from a set of possible different values, each with an associated probability (likelihood). An event is a set of outcomes of an experiment, to which a probability is assigned. For example, suppose you have an exam and you are not so well prepared. There are 10 possible questions, but you only had time to prepare for 8. If 5 questions are given, what chances do you have to be familiar with all five? The number of questions that you are familiar with is now a random variable and the case where "you are familiar with 4 out of 5 questions" is an event. For a great introductory book to Probability Theory, the reader is referred to [43].

In the theory of algorithms, randomized algorithms are algorithms that use random numbers to make decisions during their execution. Unlike deterministic algorithms that for a fixed input always give the same output and the same running-time, a randomized algorithm behaves differently from execution to execution.

The probabilistic analysis of algorithms is an approach to estimate the complexity of an algorithm or a problem, assuming a probabilistic distribution of the set of all possible inputs. This assumption is then used to design an efficient algorithm or to derive the complexity of a known algorithm. This approach is not the same as that of randomized algorithms, but the two may be combined.

For deterministic algorithms, the most common types of complexity estimates are the average-case complexity (expected time complexity) and the almost always complexity. To obtain the average-case complexity, given an input distribution, the expected time of an algorithm is evaluated, whereas for the almost always complexity estimate, it is evaluated that the algorithm admits a given complexity estimate that almost surely holds.

For randomized algorithms, the distributions or averaging for all possible choices in randomized steps are also taken into an account, in addition to the input distributions.

A.4.1 Chernoff bounds

In probability theory, the *Chernoff bound* gives exponentially decreasing bounds on tail distributions of sums of *independent* random variables [36].

The generic Chernoff bound for a random variable X is attained by applying Markov's inequality to e^{tX}. We then get that for every $t > 0$:

$$Pr[X \geq \alpha] = Pr[e^{tX} \geq e^{t\alpha}] \leq \frac{E[e^{tX}]}{e^{t\alpha}}$$

If X is the sum of n random variables $X_1, \ldots, X_n$, we get for any $t > 0$ that:

$$Pr[X \geq \alpha] \leq e^{-t\alpha} E\left[\prod_i e^{tX_i}\right]$$

Multiplicative Chernoff Bound form. Let X be the sum of n independent random variables $X_1, \ldots, X_n$, each taking values in $\{0, 1\}$. Let $\mu = E[X]$ be the expected value of X. Then, for $0 < \delta < 1$, it holds that:

$$\begin{aligned} Pr[X \geq (1+\delta)\mu] &\leq e^{-\frac{\delta^2\mu}{3}} \\ Pr[X \leq (1-\delta)\mu] &\leq e^{-\frac{\delta^2\mu}{2}} \end{aligned}$$

Using the above bounds, we can also prove the following for $0 < \delta < 1$:

Theorem A.10.

$$Pr[X \in (1 \pm \delta)\mu] \geq 1 - 2e^{-\frac{\delta^2\mu}{3}}$$

Proof. We have:

$$\begin{aligned} Pr[|X-\mu| \geq \delta\mu] &= Pr[X-\mu \geq \delta\mu] + Pr[X-\mu \leq -\delta\mu] \\ &= Pr[X \geq (1+\delta)\mu] + Pr[X \leq (1-\delta)\mu] \\ &\leq e^{-\frac{\delta^2\mu}{3}} + e^{-\frac{\delta^2\mu}{2}} \\ &\leq e^{-\frac{\delta^2\mu}{3}} + e^{-\frac{\delta^2\mu}{3}} \\ &= 2e^{-\frac{\delta^2\mu}{3}}. \end{aligned} \tag{A.1}$$

But, we also have:

$$\begin{aligned} Pr[|X-\mu| < \delta\mu] &= Pr[-\delta\mu < X-\mu < \delta\mu] \Leftrightarrow \\ \Leftrightarrow 1 - Pr[|X-\mu| \geq \delta\mu] &= Pr[(1-\delta)\mu < X < (1+\delta)\mu] \Leftrightarrow \\ \Leftrightarrow 1 - Pr[|X-\mu| \geq \delta\mu] &= Pr[X \in (1\pm\delta)\mu]. \end{aligned} \tag{A.2}$$

Relations A.1 and A.2 prove the statement of the theorem. ■

When referring to Chernoff bounds throughout this thesis, we mean one of the three multiplicative forms of the bound described above.

A.5 Linear Programming

Linear programming (LP), also called linear optimization, is a method used to achieve the best outcome (such as the maximum profit or the lowest cost) in a mathematical model, in which the requirements are represented by linear relationships. Linear programming is a special case of mathematical optimization.

More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and inequality constraints. Its feasible region is a convex polytope, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine (linear) function defined on this polyhedron [102].

Linear programs are problems that can be expressed in canonical form as follows:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & Ax \leq b \\ \text{and} & x \geq 0, \end{array}$$

where x is the vector of variables that need to be determined and c and b are vectors of the known coefficients.

The linear programming problem was first shown to be solvable in polynomial time by Leonid Khachiyan in 1979 with the introduction of the ellipsoid method, but a larger theoretical and practical breakthrough in the field came in 1984 when Narendra Karmarkar introduced a new interior-point method (projective method) for solving linear-programming problems (interior-point methods in general move through the interior of the feasible region).

For a detailed treatment of linear programming, the reader is referred to [31].

Bibliography

[1] Eric Aaron, Danny Krizanc, and Elliot Meyerson. DMVP: foremost waypoint coverage of time-varying graphs. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science - 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, volume 8747 of *Lecture Notes in Computer Science*, pages 29–41. Springer, 2014. ISBN 978-3-319-12339-4. doi: 10.1007/978-3-319-12340-0_3. URL `http://dx.doi.org/10.1007/978-3-319-12340-0_3`.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. ISBN 0-13-617549-X.

[3] Eleni C. Akrida and Paul G. Spirakis. On verifying and maintaining connectivity of interval temporal networks. In Prosenjit Bose, Leszek Antoni Gasieniec, Kay Römer, and Roger Wattenhofer, editors, *Algorithms for Sensor Systems - 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2015, Patras, Greece, September 17-18, 2015, Revised Selected Papers*, volume 9536 of *Lecture Notes in Computer Science*, pages 142–154. Springer, 2015. ISBN 978-3-319-28471-2. doi: 10.1007/978-3-319-28472-9_11. URL `http://dx.doi.org/10.1007/978-3-319-28472-9_11`.

[4] Eleni C. Akrida, Leszek Gąsieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: Diameter and connectivity. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2014.

[5] Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Designing and testing temporally connected graphs. *CoRR*, abs/1502.04579, 2015. URL `http://arxiv.org/abs/1502.04579`.

[6] Eleni C. Akrida, Leszek Gąsieniec, George B. Mertzios, and Paul G. Spirakis. On temporally connected graphs of small cost. In *Proceedings of the 13th Workshop on Approximation and Online Algorithms (WAOA)*, 2015.

[7] Eleni C. Akrida, Jurek Czyzowicz, Leszek Gasieniec, Lukasz Kuszner, and Paul G. Spirakis. Flows in temporal networks. *CoRR*, abs/1606.01091, 2016. URL `http://arxiv.org/abs/1606.01091`.

[8] Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *J. Parallel Distrib. Comput.*, 87:109–120, 2016. doi: 10.1016/j.jpdc.2015.10.002. URL `http://dx.doi.org/10.1016/j.jpdc.2015.10.002`.

[9] Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. In *Proceedings of the Third Italian Conference on Algorithms and Complexity*, CIAC '97, pages 288–298, London, UK, UK, 1997. Springer-Verlag. ISBN 3-540-62592-5. URL `http://dl.acm.org/citation.cfm?id=648256.752884`.

[10] Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\varepsilon)$-approximate flow sparsifiers. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 279–293. SIAM, 2014. ISBN 978-1-61197-338-9. doi: 10.1137/1.9781611973402.20. URL `http://dx.doi.org/10.1137/1.9781611973402.20`.

[11] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *J. Comput. Syst. Sci.*, 18(2):155–193, 1979. doi: 10.1016/0022-0000(79)90045-X. URL `http://dx.doi.org/10.1016/0022-0000(79)90045-X`.

[12] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, 2006.

[13] Jay E. Aronson. A survey of dynamic network flows. *Ann. Oper. Res.*, 20 (1-4):1–66, August 1989. ISSN 0254-5330. doi: 10.1007/BF02216922. URL `http://dx.doi.org/10.1007/BF02216922`.

[14] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004. ISBN 0-471-453242.

[15] Chen Avin, Michal Koucký, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 121–132, 2008.

[16] Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Yessin M. Neggaz. Testing temporal connectivity in sparse dynamic graphs. *CoRR*, abs/1404.7634, 2014. URL `http://arxiv.org/abs/1404.7634`.

[17] Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 47–58. SIAM, 2015. ISBN 978-1-61197-374-7. doi: 10.1137/1.9781611973730.5. URL `http://dx.doi.org/10.1137/1.9781611973730.5`.

[18] Nadine Baumann and Martin Skutella. Earliest arrival flows with multiple sources. *Math. Oper. Res.*, 34(2):499–512, 2009. doi: 10.1287/moor.1090.0382. URL `http://dx.doi.org/10.1287/moor.1090.0382`.

[19] Petra Berenbrink, Robert Elsässer, and Tom Friedetzky. Efficient randomised broadcasting in random regular networks with applications in peer-to-peer systems. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 155–164, 2008.

[20] Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996.

[21] Sandeep Bhadra and Afonso Ferreira. Computing multicast trees in dynamic networks and the complexity of connected components in evolving graphs. *J. Internet Services and Applications*, 3(3):269–275, 2012.

doi: 10.1007/s13174-012-0073-z. URL `http://dx.doi.org/10.1007/s13174-012-0073-z`.

[22] Natashia Boland, Thomas Kalinowski, Hamish Waterer, and Lanbo Zheng. Scheduling arc maintenance jobs in a network to maximize total flow over time. *Discrete Applied Mathematics*, 163, Part 1:34 – 52, 2014. ISSN 0166-218X. doi: http://dx.doi.org/10.1016/j.dam.2012.05.027. URL `http://www.sciencedirect.com/science/article/pii/S0166218X12002338`. Matheuristics 2010.

[23] Bela Bollobás. *Modern Graph Theory.* Springer, 1998.

[24] Bela Bollobás. *Random Graphs, 2nd Edition.* Cambridge University Press, 2001.

[25] Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[26] Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics. Technical report, Commissioned by Defense Research and Development Canada (DRDC), April 2013. URL `https://hal.archives-ouvertes.fr/hal-00865762`.

[27] Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Problems, Analysis, and Algorithmic Tools. Technical report, Commissioned by Defense Research and Development Canada (DRDC), April 2013. URL `https://hal.archives-ouvertes.fr/hal-00865764`.

[28] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Measuring temporal lags in delay-tolerant networks. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 209–218, May 2011. doi: 10.1109/IPDPS.2011.29.

[29] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.

[30] Augustin Chaintreau, Abderrahmen Mtibaa, Laurent Massoulié, and Christophe Diot. The diameter of opportunistic mobile networks. In Jim

Kurose and Henning Schulzrinne, editors, *Proceedings of the 2007 ACM Conference on Emerging Network Experiment and Technology, CoNEXT 2007, New York, NY, USA, December 10-13, 2007*, page 12. ACM, 2007. ISBN 978-1-59593-770-4. doi: 10.1145/1364654.1364670. URL `http://doi.acm.org/10.1145/1364654.1364670`.

[31] Vasek Chvátal. *Linear programming.* A Series of books in the mathematical sciences. Freeman, New York (N. Y.), 1983. ISBN 0-7167-1195-8. URL `http://opac.inria.fr/record=b1104676`. Réimpressions : 1999, 2000, 2002.

[32] Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time of edge-markovian evolving graphs. *SIAM Journal on Discrete Mathematics (SIDMA)*, 24(4):1694–1712, 2010.

[33] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001. ISBN 0-262-03293-7.

[34] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[35] Camil Demetrescu, Irene Finocchi, and Giuseppe F. Italiano. Dynamic graphs. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications.* Chapman and Hall/CRC, 2004. ISBN 978-1-58488-435-4. doi: 10.1201/9781420035179.ch36. URL `http://dx.doi.org/10.1201/9781420035179.ch36`.

[36] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 1st edition, 2009. ISBN 0521884276, 9780521884273.

[37] Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. On the complexity of information spreading in dynamic networks. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 717–736, 2013.

[38] Friedrich Eisenbrand, Andreas Karrenbauer, Martin Skutella, and Chihao Xu. Multiline addressing by network flow. *Algorithmica*, 53(4):583–596, 2009. doi: 10.1007/s00453-008-9252-5. URL `http://dx.doi.org/10.1007/s00453-008-9252-5`.

[39] Khaled M. Elbassioni, Naveen Garg, Divya Gupta, Amit Kumar, Vishal Narula, and Arindam Pal. Approximation algorithms for the unsplittable flow problem on paths and trees. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 267–275. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. ISBN 978-3-939897-47-7. doi: 10.4230/LIPIcs.FSTTCS.2012.267. URL `http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2012.267`.

[40] Robert Elsässer. On the communication complexity of randomized broadcasting in random-like graphs. In *Proceedings of the 18th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 148–157, 2006.

[41] Robert Elsässer and Thomas Sauerwald. The power of memory in randomized broadcasting. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 218–227, 2008.

[42] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 444–455. Springer, 2015. ISBN 978-3-662-47671-0. doi: 10.1007/978-3-662-47672-7_36. URL `http://dx.doi.org/10.1007/978-3-662-47672-7_36`.

[43] William Feller. *An Introduction to Probability Theory and Its Applications.* Number v. 1 in An Introduction to Probability Theory and Its Applications. Wiley, 1967. URL `https://books.google.co.uk/books?id=jbkdAQAAMAAJ`.

[44] Afonso Ferreira. Building a reference combinatorial model for manets. *IEEE Network*, 18(5):24–29, 2004. doi: 10.1109/MNET.2004.1337732. URL `http://dx.doi.org/10.1109/MNET.2004.1337732`.

[45] Lisa Fleischer and Martin Skutella. Quickest flows over time. *SIAM J. Comput.*, 36(6):1600–1630, 2007. doi: 10.1137/S0097539703427215. URL `http://dx.doi.org/10.1137/S0097539703427215`.

[46] Lisa Fleischer and Éva Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23(3-5):71–80, 1998.

[47] Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 534–543. Springer, 2009. ISBN 978-3-642-10630-9. doi: 10.1007/978-3-642-10631-6_55. URL `http://dx.doi.org/10.1007/978-3-642-10631-6_55`.

[48] Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010. ISBN 0691146675, 9780691146676.

[49] Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. *Theor. Comput. Sci.*, 410(36):3305–3326, 2009. doi: 10.1016/j.tcs.2008.01.004. URL `http://dx.doi.org/10.1016/j.tcs.2008.01.004`.

[50] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 252–257, New York, NY, USA, 1983. ACM. ISBN 0-89791-099-0. doi: 10.1145/800061.808754. URL `http://doi.acm.org/10.1145/800061.808754`.

[51] Michael L. Fredman and Monika Rauch Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22:351–362.

[52] Alan.M. Frieze and Geoffrey.R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.

[53] Cyril Gavoille, David Peleg, Stephane Perennes, and Ran Raz. Distance labeling in graphs. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 210–219, 2001.

[54] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow: Extended abstract. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 81–90. ACM, 2015. ISBN 978-1-4503-3617-8. doi: 10.1145/2767386.2767440. URL `http://doi.acm.org/10.1145/2767386.2767440`.

[55] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 550–559. IEEE Computer Society, 2011. ISBN 978-1-4577-1843-4. doi: 10.1109/FOCS.2011.80. URL `http://dx.doi.org/10.1109/FOCS.2011.80`.

[56] Martin Groß and Martin Skutella. Generalized maximum flows over time. In *Approximation and Online Algorithms - 9th International Workshop, WAOA 2011, Saarbrücken, Germany, September 8-9, 2011, Revised Selected Papers*, pages 247–260, 2011. doi: 10.1007/978-3-642-29116-6_21. URL `http://dx.doi.org/10.1007/978-3-642-29116-6_21`.

[57] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. *SIAM J. Comput.*, 41(6):1649–1672, 2012. doi: 10.1137/09076725X. URL `http://dx.doi.org/10.1137/09076725X`.

[58] Mohammad Taghi Hajiaghayi and Harald Räcke. An o(sqrt(n))-approximation algorithm for directed sparsest cut. *Inf. Process. Lett.*, 97 (4):156–160, 2006. doi: 10.1016/j.ipl.2005.10.005. URL `http://dx.doi.org/10.1016/j.ipl.2005.10.005`.

[59] Horst W. Hamacher and Stevanus A. Tjandra. Earliest arrival flows with time-dependent data. Tech. Rep. 88, Fachbereich Mathematik – TU Kaiserslautern, 2003. URL `https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/1449`.

[60] Frank Harary. *Graph Theory*. Addison-Wesley series in mathematics. Addison-Wesley Publishing Company, 1969. ISBN 9780201410334. URL `https://books.google.co.uk/books?id=9nOljWrLzAAC`.

[61] Monika Rauch Henzinger and Valerie King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 519–527. ACM, 1995. ISBN 0-89791-718-9. doi: 10.1145/225058.225269. URL `http://doi.acm.org/10.1145/225058.225269`.

[62] Monika Rauch Henzinger and Mikkel Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. *Random Struct. Algorithms*, 11(4):369–379, 1997. doi: 10.1002/(SICI)1098-2418(199712)11:4<369::AID-RSA5>3.0.CO;2-X. URL `http://dx.doi.org/10.1002/(SICI)1098-2418(199712)11:4<369::AID-RSA5>3.0.CO;2-X`.

[63] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 79–89. ACM, 1998. ISBN 0-89791-962-9. doi: 10.1145/276698.276715. URL `http://doi.acm.org/10.1145/276698.276715`.

[64] Bruce Hoppe. Phd thesis: Efficient dynamic network flow algorithms, 1995.

[65] Bruce Hoppe and Éva Tardos. The quickest transshipment problem. *Math. Oper. Res.*, 25(1):36–62, 2000. doi: 10.1287/moor.25.1.36.15211. URL `http://dx.doi.org/10.1287/moor.25.1.36.15211`.

[66] Jan-Philipp W. Kappmeier. *Generalizations of Flows over Time with Applications in Evacuation Optimization.* epubli GmbH, 2015. ISBN 978-3-7375-3238-9. URL `http://www.epubli.de/shop/isbn/9783737532389`.

[67] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.

[68] Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing*, 34(1): 23–40, 2004.

[69] David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, pages 504–513, 2000.

[70] Valerie King. Fully dynamic connectivity. In *Encyclopedia of Algorithms*, pages 792–793. 2016. doi: 10.1007/978-1-4939-2864-4_152. URL `http://dx.doi.org/10.1007/978-1-4939-2864-4_152`.

[71] Jon M. Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30(1):191–217, 2000. doi: 10.1137/S0097539797329142. URL `http://dx.doi.org/10.1137/S0097539797329142`.

[72] Bettina Klinz and Gerhard J. Woeginger. One, two, three, many, or: complexity aspects of dynamic network flows with dedicated arcs. *Oper. Res. Lett.*, 22(4-5):119–127, 1998. doi: 10.1016/S0167-6377(98)00009-1. URL `http://dx.doi.org/10.1016/S0167-6377(98)00009-1`.

[73] Ronald Koch, Ebrahim Nasrabadi, and Martin Skutella. Continuous and discrete flows over time - A general model based on measure theory. *Math. Meth. of OR*, 73(3):301–337, 2011. doi: 10.1007/s00186-011-0357-2. URL `http://dx.doi.org/10.1007/s00186-011-0357-2`.

[74] Vassilis Kostakos. Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023, 2009. URL `http://EconPapers.repec.org/RePEc:eee:phsmap:v:388:y:2009:i:6:p:1007-1023`.

[75] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 513–522, 2010.

[76] Jakub Lacki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 599–608. IEEE Computer Society, 2012. ISBN 978-1-4673-4383-1. doi: 10.1109/FOCS.2012.66. URL `http://dx.doi.org/10.1109/FOCS.2012.66`.

[77] Lap Chi Lau and Mohit Singh. Additive approximation for bounded degree survivable network design. *SIAM J. Comput.*, 42(6):2217–2242, 2013. doi: 10.1137/110854461. URL `http://dx.doi.org/10.1137/110854461`.

[78] Lap Chi Lau, Joseph Naor, Mohammad R. Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM J. Comput.*, 39(3):1062–1087, 2009. doi: 10.1137/070700620. URL `http://dx.doi.org/10.1137/070700620`.

[79] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007. doi: 10.1145/1217299.1217301. URL `http://doi.acm.org/10.1145/1217299.1217301`.

[80] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 245–254. IEEE Computer Society, 2010. ISBN 978-0-7695-4244-7. doi: 10.1109/FOCS.2010.30. URL `http://dx.doi.org/10.1109/FOCS.2010.30`.

[81] George B. Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP), Part II*, pages 657–668, 2013.

[82] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016. doi: 10.1080/15427951.2016.1177801. URL `http://dx.doi.org/10.1080/15427951.2016.1177801`.

[83] Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 553–564. Springer, 2014. ISBN 978-3-662-44464-1. doi: 10.1007/978-3-662-44465-8_47. URL `http://dx.doi.org/10.1007/978-3-662-44465-8_47`.

[84] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theoretical Computer Science*, 412(22):2434–2450, 2011.

[85] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. *New Models for Population Protocols.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.

[86] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. In *OPODIS*, pages 269–283, 2012.

[87] Michael Molloy and Bruce Reed. *Graph colouring and the probabilistic method*, volume 23 of *Algorithms and Combinatorics.* Springer, 2002.

[88] Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. Graph Metrics for Temporal Networks. In Petter Holme and Jari Saramäki, editors, *Temporal Networks*, Understanding Complex Systems, pages 15–40. Springer, May 2013.

[89] Regina O'Dell and Roger Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the 2005 joint workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 104–110, 2005.

[90] James B. Orlin. Max flows in o(nm) time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. ISBN 978-1-4503-2029-0. doi: 10.1145/2488608.2488705. URL `http://doi.acm.org/10.1145/2488608.2488705`.

[91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991. doi: 10.1016/0022-0000(91)90023-X. URL `http://dx.doi.org/10.1016/0022-0000(91)90023-X`.

[92] Christos M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821.

[93] Boris Pittel. Linear probing: The probable largest search time grows logarithmically with the number of records. *J. Algorithms*, 8(2):236–249, 1987.

[94] Warren B Powell, Patrick Jaillet, and Amedeo Odoni. Stochastic and dynamic networks and routing. *Handbooks in operations research and management science*, 8:141–295, 1995.

[95] Christian Scheideler. Models and techniques for communication in dynamic networks. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2285, pages 27–49, 2002.

[96] Martin Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization, Bonn Workshop on Combinatorial Optimization, November 3-7, 2008, Bonn, Germany*, pages 451–482, 2008. doi: 10.1007/978-3-540-76796-1_21. URL `http://dx.doi.org/10.1007/978-3-540-76796-1_21`.

[97] John Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *Computer Communication Review*, 40(1):118–124, 2010. doi: 10.1145/1672308.1672329. URL `http://doi.acm.org/10.1145/1672308.1672329`.

[98] John Kit Tang. *Temporal network metrics and their application to real world networks*. PhD thesis, Citeseer, 2011.

[99] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350. ACM, 2000. ISBN 1-58113-184-4. doi: 10.1145/335305.335345. URL `http://doi.acm.org/10.1145/335305.335345`.

[100] John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In Özgür B. Akan, Eylem Ekici, Lili Qiu, and Alex C. Snoeren, editors, *The 18th Annual International Conference on Mobile Computing and Networking, Mobicom'12, Istanbul, Turkey, August 22-26, 2012*, pages 377–388. ACM, 2012. ISBN 978-1-4503-1159-5. doi: 10.1145/2348543.2348589. URL `http://doi.acm.org/10.1145/2348543.2348589`.

[101] Wikipedia. Sharp-p-complete — wikipedia, the free encyclopedia, 2015. URL `https://en.wikipedia.org/w/index.php?title=Sharp-P-complete&oldid=686300526`. [Online; accessed 18-October-2015].

[102] Wikipedia. Linear programming — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Linear_programming&oldid=746128941. [Online; accessed 25-October-2016].

[103] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, May 2014. ISSN 2150-8097. doi: 10.14778/2732939.2732945. URL http://dx.doi.org/10.14778/2732939.2732945.

[104] Huanhuan Wu, James Cheng, Yi Lu, Yiping Ke, Yuzhen Huang, Da Yan, and Hejun Wu. Core decomposition in large temporal graphs. In *Big Data*, pages 649–658. IEEE, 2015. ISBN 978-1-4799-9926-2. URL http://dblp.uni-trier.de/db/conf/bigdataconf/bigdataconf2015.html#WuCLKHYW15.