Exploring Temporal Cycles and Grids

Shadi Taghian Alamouti

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

December 2020

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Shadi Taghian Alamouti

Entitled: Exploring Temporal Cycles and Grids

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Thomas Fevens

_____ Examiner
Dr. Lata Narayanan

_____ Examiner
Dr. Thomas Fevens

_____ Thesis Supervisor(s)
Dr. Hovhannes Harutyunyan

_____ Thesis Supervisor(s)
Dr. Denis Pankratov

Approved by _____
Dr. Hovhannes Harutyunyan Chair of Department or Graduate Program Director

_____
Dean Dr. Mourad Debbabi

Abstract

Exploring Temporal Cycles and Grids

Shadi Taghian Alamouti

A temporal graph is a graph in which the edge set can change from time to time. The temporal graph exploration problem, TEXP, is the problem of computing a foremost exploration schedule for a temporal graph, i.e., a temporal walk that starts at a given start node, visits all nodes of the graph, and has the smallest arrival time.

In this work, we consider undirected temporal graphs that are connected at each time step. We present an overview of some known results regarding exploration of temporal cycles and grids, as well as new contributions to the theory of TEXP of cycles and grids. Our results can be grouped into two sets. The first set of results concerns TEXP of cycles. In particular, we positively resolve the conjecture of Erlebach et al. stating that temporal cycles of size n with constantly many chords can be explored in $O(n)$ time steps. Moreover, we design a polynomial time dynamic programming algorithm for computing an optimal temporal walk for TEXP on cycles from any start node. We implemented this algorithm and performed a limited empirical evaluation, results of which are also reported in this thesis. Our second set of results extends the algorithm of Erlebach et al. for the exploration of $2 \times n$ temporal grids to modified temporal grids that allow some neighboring edges to cross each other.

Acknowledgments

I would like to express my deepest appreciation to my co-supervisors Dr. Harutyunyan and Dr. Pankratov for their invaluable insight and advice, as well as guidance. I would also like to thank my husband for being by my side throughout this work every moment. I am also thankful to my mom, dad and sisters for their never ending support.

This thesis is dedicated to the souls of all 176 passengers of Flight PS752, who were shot down shortly after takeoff on January 8th 2020.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Preliminaries

The conception and development of graph theory are among the most important achievements in combinatorics and mathematics at large. Applications of graph theory are numerous and extend to many related (and not so related) fields, including computer science, chemistry, biology, etc. One would be hard pressed to name an area of science, mathematics, and engineering that has *not* benefited from the development of graph theory.

A graph is used to mathematically model objects (also called vertices or nodes) and pairwise relations between the objects (also called edges, links, or arcs). A graph can be used to model transportation networks, electrical grids, social networks, to name a few real-life scenarios. As such, the question of exploration of graphs is of paramount interest. One can formalize the exploration question in several ways. In one formalization, the goal is to visit every edge exactly once. This problem is known as the Eulerian path/cycle problem. A particular instance of this problem, which is of historical importance, is known as the seven bridges of Königsberg problem. The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River and included two large islands — Kneiphof and Lomse — which were connected to each other, and to the two mainland portions of the city, by seven bridges. The problem was to devise a walk through the city that would cross each of those bridges once and only once. In 1736, Leonhard Euler [10] proved that the problem has no solution. The difficulty he faced was the development of a suitable technique of analysis and of subsequent tests that established this assertion with mathematical rigor. This laid the foundations of graph theory and prefigured the

idea of topology [26]. The general version of this problem was named after Euler in his honour. This problem has efficient algorithmic solutions.

Another way to formalize the graph exploration problem is to request a path (if it exists) that visits every vertex (instead of each edge) exactly once. This is known as the Hamiltonian path problem. If a Hamiltonian path exists whose endpoints are adjacent, then the resulting graph cycle is called a Hamiltonian cycle. Hamiltonian paths and cycles are named after William Rowan Hamilton, who in 1857 invented the icosian game, now also known as Hamilton's puzzle, which involves finding a Hamiltonian cycle in the edge graph of the dodecahedron. In geometry, a dodecahedron is any polyhedron with twelve flat faces. The most familiar dodecahedron is the regular dodecahedron with regular pentagons as faces, which is a Platonic solid. There are also three regular star dodecahedra, which are constructed as stellations of the convex form. Hamilton solved this problem using the icosian calculus, an algebraic structure based on roots of unity with many similarities to the quaternions (also invented by Hamilton). This solution does not generalize to arbitrary graphs. Unlike Eulerian path problem, Hamiltonian path problem is **NP**-hard and thus does not admit efficient algorithms unless $\mathbf{P} = \mathbf{NP}$. The weighted version of the Hamiltonian cycle problem is known as the travelling salesman problem.

This thesis is concerned with a graph exploration problem most closely related to the Hamiltonian path problem, but in a temporal setting and allowing for multiple visits to the same vertex. The motivation for studying the temporal setting is that many networks are not static and change over time. For example, in a transportation network, the connections might function only at certain times. Moreover, connections in social networks keep changing as different subnetworks are created and removed. Links in wired or wireless networks may change dynamically. As another example, consider the following problem, which is a modified version of the traveling salesman problem. A tourist travels to a country and wants to visit all k touristic cities using trains in a given period of time. Each train has a traveling schedule of its own. The tourist must choose the best schedule in order to visit all cities since the trains are not available at all times. This problem is an example of exploration in a temporal graph. Such dynamically changing networks have been modelled in various ways and studied in the context of faulty networks, scheduled networks, time-varying networks, distributed algorithms, etc. See [9] and references therein. While different models

tend to have slightly different emphasis (for example, online algorithms tend to be concerned with irrevocable decisions, dynamic algorithms tend to be concerned with data structures, streaming algorithms tend to be concerned with memory requirements), we focus on the model called "temporal graphs", where the schedule of how the graph changes with time is known in advance and one is interested in designing efficient offline algorithms for various combinatorial problems.

More specifically, in this work, we discuss the temporal graph exploration problem TEXP. The temporal graph exploration problem, TEXP, is the problem of computing a foremost exploration schedule for a temporal graph, i.e., a temporal walk that starts at a given start node, visits all nodes of the graph (repetition is allowed), and has the smallest arrival time. Such a walk needs to be consistent with the schedule of graph changes, i.e., it can use an edge at a particular time only if that edge is present at that time. We often visualize such a walk as being carried out by a single agent. As a sub-routine or a proof-technique, we also sometimes consider a multi-agent variant k-TEXP of TEXP in which there are k agents initially located at the start vertex. In this thesis, k-TEXP is used only because it is easier to design exploration schedules with multiple agents and we can use various tricks to convert such schedules into a schedule for a single agent. We focus on the TEXP of temporal cycles and temporal grids. The following is the list of our contributions presented in this thesis:

- Erlebach et al. [9] proved that a connected temporal cycle (even with one chord) can be explored by a single agent in $O(n)$ time steps. They conjectured that the same bound should hold for connected temporal cycles with constantly many chords. We prove this conjecture. This is one of the more technical of our contributions.

- While Erlebach et al. gave careful proofs of tight bounds on the exploration time of connected temporal cycles, they only sketched how to compute an optimal exploration time for a given start node. We give a different polynomial time algorithm based on dynamic programming that computes an optimal exploration time for connected temporal cycles.

- We implemented and performed a limited empirical evaluation of our dynamic programming algorithm from the previous point.

- We give alternative proofs to similar bounds as those of Erlebach et al. on the worst-case exploration time of temporal cycles.
- We adapt the algorithm of Erlebach et al. for the exploration of $2 \times n$ temporal grids to other related topologies, such as modified grids that allow some neighboring edges to cross each other.

Organization. The rest of this thesis is organized as follows. In the remainder of this chapter, we present definitions and preliminaries: main properties, lemmas, and theorems that are needed to formally state the main problems of this work and to present our contributions. In Chapter 2, we review the previous literature on the topic of temporal graph exploration. Review of some more technical results concerning temporal cycles and grids is postponed to a later chapter, so that it appears closer to our related results. In Chapter 2, we also introduce and discuss the important dichotomy of temporal graph problems, namely, the algorithmic problems and the design problems. Our contributions appear in Chapter 3 grouped into two sets of results: those concerning TEXP of temporal cycles and those concerning TEXP of temporal grids. Lastly, we finish with conclusions in Chapter 4.

1.1 Preliminaries

1.1.1 Classical/Static Graphs

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. In case of undirected simple graphs, an edge $e \in E$ is a set of two vertices, i.e., $e = \{u, v\}$ where $u, v \in V$ with $u \neq v$. A set $\{u, v\}$ is used to indicate that an edge has no orientation or direction. The set of all subsets of V of size 2 is denoted by $\binom{V}{2}$ by analogy with binomial coefficients. Thus, for undirected simple graphs we have $E \subseteq \binom{V}{2}$. In case of directed simple graphs (or simply digraphs), we have $E \subseteq V \times V \setminus \{(u, u) : u \in V\}$, i.e., each edge is a pair (u, v) with $u \neq v$; the interpretation being that the edge is oriented from u to v. More generally, multi-graphs and multi-digraphs allow several parallel edges defined between a pair of vertices, as well as self-loops. In this thesis, we consider only simple graphs which allow only a single edge to be present between any two vertices and disallow self-loops. Thus, when we write "graph" we mean a simple undirected graph.

Additional information may be encoded with weights (i.e., labels) that are placed on edges and/or vertices. For example, in a graph modelling a road network of a country, the weight of the edge (C_1, C_2) can be used to represent the average time it takes to drive from city C_1 to C_2. Formally, an edge-weighted graph G comes with the weight function $w : E \to \mathbb{R}$ for real weights, but more generally the range of w could be any set. Similarly, a vertex-weighted graph with real weights comes with the weight function of the form $w : V \to \mathbb{R}$. When we say "weighted graph" without prefix "edge-" or "vertex-" we mean "edge-weighted graph". For a more detailed introduction to classical graph theory, an interested reader is referred to [28] and [7].

1.1.2 Three Equivalent Models of Temporal Graphs

Temporal graphs (also known as dynamic[1], evolving [11], or time-varying [12], [4] graphs) can be informally described as graphs that change with time. In this thesis, we focus on temporal graphs that change in discrete time steps (as opposed to continuous time setting). There are several equivalent ways of formally modelling temporal graphs. The first way is to represent a temporal graph by a sequence of graphs – one graph per time step. More formally, we have:

Definition 1.1.1. *A temporal graph $\mathcal{G}$ with vertex set V and lifetime (or age) L is given by a sequence of graphs $(G_i), 0 \leq i \leq L$ with $G_i = (V, E_i)$. We refer to $i, 0 \leq i \leq L$, as time i or step i. The graph $G = (V, E)$ with $E = \bigcup_{0 \leq i \leq L} E_i$ is called the underlying graph of $\mathcal{G}$.*

If the underlying graph is X, then we call the temporal graph a *temporal* X or a *temporal realization* of X. For example, a temporal cycle is a temporal graph whose underlying graph is a cycle. We use the term "*static*" to refer to classical graphs. This is plausible as the opposite of "*dynamic*" that is sometimes used for temporal graphs as mentioned above.

Another way to model temporal graphs is by using weighted/labelled graphs, where the label of the edge is the set of times when the edge is present. More formally, we have:

[1] One should be careful with the term "dynamic graph" and infer the meaning from the context, as it is also used to refer to the area of dynamic algorithms for graphs, where the objective is to design data structures with efficient update operations. This usage of the term is different from "temporal graphs." As such we shall try to avoid using the term "dynamic graphs" as much as possible in the remainder of the thesis.

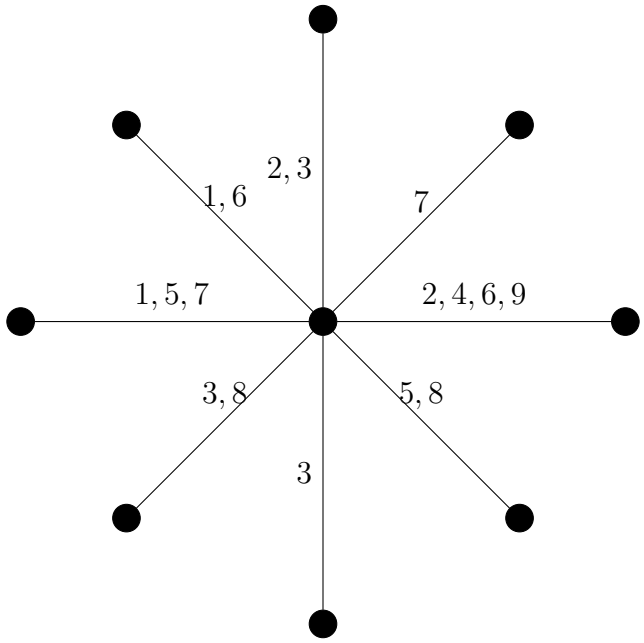

Figure 1: A temporal star: The labels on each edge shows the time step in which the edge is present.

Definition 1.1.2. *Consider an underlying static graph $G = (V, E)$ together with a labeling $\lambda : E \to 2^{\mathbb{N}}$ of G assigning to every edge of G a (possibly empty) set of natural numbers, called labels. Then the temporal graph of G with respect to λ is denoted by $\lambda(G)$.*

This second model of temporal graphs is particularly well suited for illustrations. An example is given in Figure 1. This notation is also useful when we want to study the properties of the labels of the temporal graphs. For example, the multiset of all labels of $\lambda(G)$ can be denoted by $\lambda(E)$, their cardinality is defined as $|\lambda(E)| = \sum_{e \in E} |\lambda(e)|$, and the maximum and minimum labels assigned to the whole temporal graph as $\lambda_{\max} = \max \lambda(E)$ and $\lambda_{\min} = \min \lambda(E)$, respectively. Observe that in this model the age (or the lifetime) of a temporal graph $\lambda(G)$ as $\alpha(\lambda) = \lambda_{\max} - \lambda_{\min} + 1$ (or simply α when G is clear from the context). Note that in the case $\lambda_{\min} = 1$ then we have $\alpha(\lambda) = \lambda_{\max}$, and the definition of the age (or the lifetime) matches that of the first model above.

Yet another, often convenient, formalization of a temporal graph $\mathcal{G}$ is as follows:

Definition 1.1.3. *A temporal graph $\mathcal{G}$ is an ordered pair of disjoint sets (V, A) such that $A \subseteq \binom{V}{2} \times \mathbb{N}$ in case of a simple undirected graph and with $\binom{V}{2}$ replaced by*

$V^2 \backslash \{(u,u) : u \in V\}$ *in case of a simple digraph.*

The set A in the previous definition is called the set of time-edges. A can also be used to refer to the structure of the temporal graph at a particular time. In particular, $A(t) = \{e : (e,t) \in A\}$ is the (possibly empty) set of all edges that appear in the temporal graph at time t. In turn, $A(t)$ can be used to define a snapshot of the temporal graph $\mathcal{G}$ at time t, which is usually called the t^{th} instance of $\mathcal{G}$, and is the static graph $\mathcal{G}(t) = (V, A(t))$. So, it becomes evident that this formalization is equivalent to the first formalization in terms of a sequence of static graphs $(G_1, G_2, \ldots, G_{\lambda_{max}})$.

Finally, it is typically very useful to expand in time the whole temporal graph and obtain an equivalent static graph without losing any information. The reason for doing this is mainly because static graphs are much better understood, and there is a rich set of well-established tools and techniques for them. So, a common approach to make initial progress on a problem concerning temporal graphs is to first express the given temporal graph as a static graph and then try to apply or adjust one of the existing tools that work on static graphs.

Formally, the static expansion of a temporal graph $\mathcal{G} = (V, A)$ is a directed acyclic graph (DAG) $H = (S, E)$ defined as follows.

Definition 1.1.4. [22] *If* $V = \{u_1, u_2, \ldots, u_n\}$ *then* $S = \{u_{ij} : \lambda_{min} - 1 \leq i \leq \lambda_{max}, 1 \leq j \leq n\}$ *and* $E = \{(u_{(i-1)j}, u_{ij'}) : \lambda_{min} \leq i \leq \lambda_{max}$ *and* $j = j'$ *or* $(u_j, u_{j'}) \in A(i)\}$.

In words, for every discrete moment we create a copy of V representing the instance of the nodes at that time (called time-nodes). We may imagine the moments as levels or rows from top to bottom, every level containing a copy of V. Then we add outgoing edges from time-nodes of one level only to time-nodes of the level below it. In particular, we connect a time-node $u_{(i-1)j}$ to its own subsequent copy u_{ij} and to every time-node $u_{ij'}$ s.t. $(u_j, u_{j'})$ is an edge of the temporal graph at time i. Observe that the above construction includes all possible vertical edges from a node to its own subsequent instance. These edges express the fact that nodes are usually not oblivious and can preserve themselves on history in time (modeled like propagating information to themselves). Nevertheless, depending on the application, these edges may sometimes be omitted [22].

We shall use all these notations interchangeably depending on which notation is more suitable for the given circumstances.

1.1.3 Temporal Graph Problems

In spite of strong connections between temporal graphs and static graphs explained in Section 1.1.2, the time aspects of temporal graph/networks give rise to a whole new set of problems and challenges. In particular, it is not yet fully understood how the complexity of combinatorial optimization problems is affected by introducing to them a notion of time. In the past few years intensive research efforts have been dedicated to numerous areas of dynamic networks and temporal graphs [22, 1, 19, 9]. This has resulted in obtaining a lot of insights regarding the new set of problems. These studies and research have mainly been in:

(a) the study of communication in highly dynamic networks, e.g., broadcasting and routing in delay-tolerant networks.

(b) the exploitation of passive mobility, e.g., the opportunistic use of transportation networks.

(c) the analysis of complex real-world networks ranging from neuroscience or biology to transportation systems or social studies, e.g., the characterization of the interaction patterns emerging in a social network.

An interesting fact about these problems is that these concepts are highly related. As a matter of fact, in several cases, differently named concepts identified by different researchers are actually one and the same concept [5]. For example, the concept of temporal distance, formalized in [30], is the same as reachability time [14], information latency [17], and temporal proximity [18]; similarly, the concept of journey [30] has been coined schedule-conforming path [3], time-respecting path [14, 16], and temporal path [6, 27]. Hence, the concepts discovered in these investigations can be viewed as parts of the same conceptual universe; and the formalisms proposed so far to express them as fragments of a larger formal description of this universe.

Among the other few things that we know is that the *max-flow min-cut* theorem holds with unit capacities for time-respecting paths [3]. Additionally, Kempe et al. [16] proved that, in temporal graphs, the classical formulation of Menger's theorem

is violated, and the computation of the number of node-disjoint $s - z$ paths becomes **NP**-complete.

Another important problem is that of designing an efficient temporal graph that satisfies the given requirements given. This problem has been studied in [21], where several interesting cost minimization parameters for optimal temporal network design has been introduced. One of the parameters is the *temporality* of a graph G, in which the goal is to create a temporal version of G minimizing the maximum number of labels of an edge, and the other is the temporal cost of G, in which the goal is to minimize the total number of labels used. Optimization of these parameters is performed subject to some connectivity constraints.

Several upper bounds and lower bounds for temporality of some basic graph families have been provided, For instance, the bounds for rings, directed acyclic graphs, and trees have been calculated. In [21], the authors have also provided a trade-off between the temporality and the maximum labels of rings. Furthermore, they gave a method for computing a lower bound of the temporality of an arbitrary graph with respect to (abbreviated w.r.t.) the constraint of preserving a time-respecting analogue of every simple path of G. Finally, they showed that computing the temporal cost w.r.t. the constraint of preserving at least one time-respecting path from u to v whenever v is reachable from u in G, is **APX**-hard [22].

Next we discuss shortest path problems in a temporal setting. Shortest path is a fundamental graph problem with numerous applications. However, the concept of classic shortest path is insufficient or even flawed in a temporal graph, as the temporal information determines the order of activities along any path. A temporal path is defined as follows:

Definition 1.1.5. *A temporal path P in a temporal graph $\mathcal{G}$ is a sequence of vertices $P = \langle v_1, v_2, \ldots, v_k, v_{k+1} \rangle$, where $\forall 1 \leq i \leq k$ the edge $(v_i, v_{i+1}) \in E_i$.*

Four types of "shortest" paths in a temporal graph have been defined. Collectively we call them minimum temporal paths, as they give the minimum value for different measures. By introducing time into a graph, the term "shortest" must be defined specifically. What do we mean by "shortest"? Is it the smallest number of edges between two nodes? Or the shortest duration of the path between two nodes? We give the formal definitions as follows:

1. ***earliest-arrival/foremost* path:** A path that gives the earliest arrival time starting from a source node s to a target node t.

2. ***latest-departure* path:** A path that gives the latest departure time starting from a source node s in order to reach a target node t by a given time.

3. ***fastest* path:** A path by which one goes from x to y with the minimum elapsed time.

4. ***shortest* path:** A path that is shortest from x to y in terms of overall traversal time needed on the edges.

It's important to note that a shortest path might not necessarily be a fastest path. For example, consider a traffic network. In such a network the shortest path from x to y may have a lot of traffic lights, whereas a highway is longer but is the fastest way to travel from x to y. Also, a fastest path might not be an earliest-arrival path either. Consider the same traffic network. Traveling from starting point x to the destination y may take 1 hour at noon due to less traffic. However, one may leave at 9 a.m., take 2 hours to travel and arrive before noon. Due to the additional temporal information, computing temporal paths and their "time-distance" poses new challenges. For example, the greedy strategy used to compute shortest paths in a static graph (e.g., by Dijkstra's algorithm) is based on the property that a subpath of a shortest path is also shortest, which is not necessarily true when computing any of the four minimum temporal paths [29].

A temporal path that visits every node of the underlying graph at least once provides an exploration of a temporal graph:

Definition 1.1.6. *Graph traversal (also known as graph exploration) refers to the process of visiting (checking and/or updating) each vertex in a graph. Such traversals are classified by the order in which the vertices are visited.*

In TEXP the goal is to find the foremost exploration path of a given temporal graph. In this work we study TEXP for various classes of graphs, such as cycles, cycles with chords, and (modified) grids. A temporal cycle is a temporal graph whose underlying graph is a cycle. A *chord* in a given cycle C is an edge between two nodes v and u, where $v, u \in V(C)$ and the edge $(v, u) \notin E(C)$. See Figure 2 for examples of temporal graphs of various classes. Another feature that helps a lot

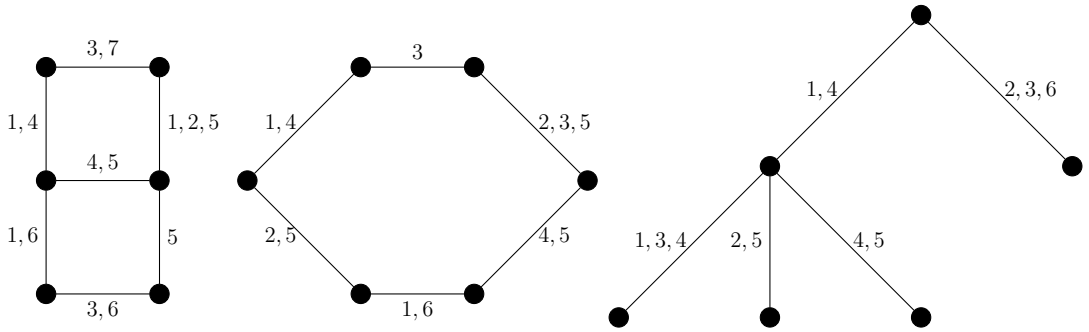

Figure 2: From left to right: a temporal grid (can also be considered a temporal cycle with 1 chord), a temporal cycle, a temporal tree.

with many temporal graph problems, specifically with temporal graph exploration, is connectivity. The concept of connectivity plays a major role when it comes to exploration. In this thesis, we only consider the temporal graphs which are connected in every time steps.

Chapter 2

Literature Review

This chapter studies and discusses the different problems on temporal graphs. Here we present both algorithmic and design problems and discuss their algorithm. Moreover, we present a few exploration problems on some families of graphs which are closely related to our main contribution. In Section 2.1, we discuss graph exploration in detail and present the very first problems regarding this matter. We continue by discussing problems on dissemination and gathering of information in temporal graphs. We introduce gossiping problem, telephone problem, as well as minimum broadcast time problem. In Section 2.3 of this chapter, we introduce random temporal graphs followed by some basic problems.

2.1 Temporal Graph Exploration

When it comes to graph exploration, the very first concepts that need to be investigated thoroughly are paths and walks.

2.1.1 Temporal Path Exploration

As is the case in static graphs, the notion of a path is one of the most central notions of a temporal graph; however, it has to be redefined to take time into account.

A *journey* (or temporal/time-respecting path) J is a temporal walk with pairwise distinct nodes. In words, a journey of $\mathcal{G}$ is a path of the underlying static graph of $\mathcal{G}$ that uses strictly increasing edge-labels. A $u-v$ journey J is called *foremost* from time $t \in \mathbb{N}$ if it departs after time t and its arrival time is minimized. As previously

defined in Chapter 1, we can give the formal definition for a temporal walk as follows.

Definition 2.1.1. [22] *A temporal (or time-respecting) walk W in a temporal graph $\mathcal{G} = (V, E)$ is an alternating sequence of nodes and times $(u_1, t_1, u_2, t_2, \ldots, u_{k-1}, t_{k-1}, u_k)$ where $(u_i u_{i+1}, t_i) \in E$, for all $1 \leq i \leq k-1$, and $t_i < t_{i+1}$, for all $1 \leq i \leq k-2$. We call $t_{k-1} - t_1 + 1$ the duration (or temporal length) of the walk W, t_1 its departure time and t_{k-1} its arrival time.*

Definition 2.1.2. [22] *The temporal distance from a node $u \in V(\mathcal{G})$ at time t to a node $v \in V(\mathcal{G})$ is defined as the duration of a foremost journey/path from u to v from time t.*

Definition 2.1.3. [22] *We say that a temporal graph $\mathcal{G} = (V, E)$ has temporal (or dynamic) diameter d, if d is the minimum integer for which it holds that the temporal distance from every time-node $(u, t) \in V \times \{0, 1, \ldots, \alpha - d\}$ to every node $v \in V$ is at most d.*

A nice property of foremost journeys is that they can be computed efficiently. In particular there is an algorithm that, given a source node $s \in V$ and a time t_{start}, computes for all $w \in V\{s\}$ a foremost $s - w$ journey from time t_{start} [21]. The running time of the algorithm is $O(n\alpha^3(\lambda) + |\lambda|)$, where n here and throughout this work denotes the number of nodes of the temporal graph. And the notations λ and α are as defined in Chapter 1. It is worth mentioning that this algorithm takes as input the whole temporal graph D. Such algorithms are known as offline algorithms in contrast to online algorithms to which the temporal graph is revealed on the fly.

The algorithm is essentially a temporal translation of the breadth-first search (BFS) algorithm (see, e.g., [7] page 531) with path length replaced by path arrival time. For every time t, the algorithm picks one after the other all nodes that have been already reached (initially only the source node s) and inspects all edges that are incident to that node at time t. If a time-edge (e, t) leads to a node w that has not yet been reached, then (e, t) is picked as an edge of a foremost journey from the source to w. This greedy algorithm is correct for the same reason that the BFS algorithm is correct. An immediate way to see this is by considering the static expansion of the temporal graph. The algorithm begins from the upper copy (i.e., at level 0) of the source in the static expansion and essentially executes the following slight variation of BFS: at step $i+1$, given the set R of already reached nodes at level i, the algorithm

first follows all vertical edges leaving R in order to reach in one step the $(i+1)$-th copy of each node in R, and then inspects all diagonal edges leaving R to discover new reachabilities.

The algorithm outputs as a foremost journey to a node u, the directed path of time-edges by which it first reached the column of u (vertical edges are interpreted as waiting on the corresponding node). The above algorithm computes a shortest path to each column of the static expansion. Correctness follows from the fact that shortest paths to columns are equivalent to foremost journeys to the nodes corresponding to the columns [22].

2.1.2 Temporal Star Exploration

An interesting family of graphs to discuss is the star family. Since there is one central node in a star that is connected to every other nodes, it can help with the exploration.

Akrida et al. introduced an interesting set of problems in [1]. They mainly focused on temporal graphs where the underlying graph is a star graph, and they consider the problem of exploring such a temporal graph starting and finishing at the center of the star. The motivation behind this is inspired by the well known *Traveling Salesman Problem (TSP)*.

However, what happens when the traveling salesman has particular temporal constraints that need to be satisfied, e.g. (s)he can only go from city A to city B on Mondays or Tuesdays, or when (s)he needs to take the train and, hence, schedule his/her visit based on the train timetables?

In particular, consider a traveling salesman who, starting from his/her home town, has to visit $n-1$ other towns via train, always returning to their own home town after visiting each city. There are trains between each town and the home town only on specific times/days, possibly different for different towns, and the salesman knows those times in advance. Can the salesman decide whether (s)he can visit all towns and return to the own home town by a certain day [1]?

Definition 2.1.4. [1] *(Temporal Star). A temporal star is a temporal graph (G_s, L) on a star graph $G_s = (V, E)$. Henceforth, we denote by c the center of G_s, i.e. the vertex of degree $n-1$.*

Definition 2.1.5. [1] *(Time edge). Let $e = (u, v)$ be an edge of the underlying graph*

of a temporal graph and consider a label $l \in \lambda(e)$. The ordered triplet (u, v, l) is called a time edge.

Definition 2.1.6. [1] *(Journey). A temporal path or journey j from a vertex u to a vertex v ($(u, v) - journey$) 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, l_k, arrival time of the journey.*

Akrida et al. identifies two problems in [1]. First they mention $StarExp(k)$. This problem answers the following question: is a temporal star (G_s, L) such that every edge has at most k labels, explorable?. The second problem that they discuss is $MaxStarExp(k)$. Given a temporal star (G_s, L) such that every edge has at most k labels, can we find a (partial) exploration of (G_s, L) of maximum size?

Theorem 2.1.1. [1] *$MaxStarExp(2)$ can be solved in $O(n \log n)$ time.*

Theorem 2.1.2. [1] *$StarExp(3)$ can be solved in $O(n \log n)$ time on instances with distinct labels.*

Corollary 2.1.1. [1] *$StarExp(3)$ can be solved in $O(n \log n)$ time on arbitrary instances.*

In order to show the hardness of these problems, Akrida et al.[1] used the problem $3SAT(3)$. This is a special case of $3SAT$. The input to this problem is a boolean formula in CNF with variables $x_1, x_2, \ldots, x_p$ and clauses $c_1, c_2, \ldots, c_q$, such that each clause has at most 3 literals, and each variable appears in at most 3 clauses. And the output to this problem is the decision on whether the formula is satisfiable.

Theorem 2.1.3. [1] *$StarExp(k)$ is NP-complete for every $k \geq 6$.*

Theorem 2.1.4. [1] *$MaxStarExp(k)$ is APX-hard, for $k \geq 6$.*

Theorem 2.1.5. [1] *Let $\epsilon_0 > 0$ be the constant such that, unless $P = NP$, there exists no polynomial-time constant-factor approximation algorithm for $Max3SAT(3)$ with approximation ratio greater than $(1 - \epsilon_0)$. Then, unless $P = NP$, there exists no polynomial-time constant-factor approximation algorithm for $MaxStarExp(k)$ with approximation ratio greater than $(1 - \frac{\epsilon_0}{19})$.*

Finally, they present some open problems. They pose a question regarding the complexity of the maximization problem MaxStarExp(3), which remains an open problem, as well as the complexity of StarExp(k) and MaxStarExp(k), for $k \in \{4, 5\}$. An interesting variation of StarExp(k) and MaxStarExp(k) is the case where the consecutive labels of every edge are λ time steps apart, for some $\lambda \in \mathbb{N}$. What is the complexity and/or best approximation factor one may hope for in this case?

2.2 Dissemination and Gathering of Information

A natural application domain of temporal graphs is that of *gossiping* and in general of *information dissemination*, mainly by a distributed set of entities (e.g., a group of people or a set of distributed processes). Two early such examples were the *telephone problem* [2] and the *minimum broadcast time problem* [25]. In both, the goal is to transmit some information to every participant of the system, while minimizing some measure of communication or time. A more modern setting, but in the same spirit, comes from the very young area of distributed computing in highly dynamic networks [24], [19], [20], [5], [23], [21].

There are n nodes. In this context, nodes represent distributed processes. Note, however, that most of the results that we will discuss, concern centralized algorithms (and in case of lower bounds, these immediately hold for distributed algorithms as well). The nodes communicate with other nodes in discrete rounds by interchanging messages. In every round, an adversary scheduler selects a set of edges between the nodes, and every node may communicate with its current neighbors, as selected by the adversary, usually by broadcasting a single message to be delivered to all its neighbors. So, the dynamic topology behaves as a discrete temporal graph where the $i-th$ instance of the graph is the topology selected by the adversary in round i. The main difference, compared to the setting of the previous sections, is that now (in all results that we will discuss in this section, apart from the last one) the topology is revealed to the algorithms in an online and totally unpredictable way. An interesting special case of temporal graphs consists of those temporal graphs that have *connected instances*.

Definition 2.2.1. [24, 19] *A temporal graph $\mathcal{D}$ is called continuously connected (also known as 1-interval connected) if $\mathcal{D}(t)$ is connected for all times $t \geq 1$.*

Such temporal graphs have some very useful properties concerning information propagation in a distributed setting, like, for example, that if all nodes broadcast in every round all information that they have heard so far, then in every round at least one more node learns something new, which implies that a piece of information can in principle be disseminated in at most $n-1$ rounds. Naturally, the problem of information dissemination becomes much more interesting and challenging if we do not allow nodes to transmit an unlimited amount of information in every round, that is, if we restrict the size of the messages that they can transmit [22].

Erlebach et al. have established some very useful lemmas and theorems in [9] that helps us in our results. The following lemma allows us to bound the time steps of a temporal walk from one vertex to another vertex in a temporal graph.

Lemma 2.2.1. [9] **(reachability)** *Let $\mathcal{G}$ be a temporal graph with vertex set V . Assume that an agent is at vertex u. Let v be another vertex and H a subset of the vertices that includes u and v and has size k. If there exists a set of $k-1$ subsequent steps such that the subgraph induced by H contains a path from u to v (which can be a different path in each step), then the agent can move from u to v in these $k-1$ steps.*

The next lemma shows that a solution to k-TEXP also yields a solution to TEXP.

Lemma 2.2.2. [9] **(multi-agent to single-agent)** *Let G be a graph with n vertices. If any temporal realization of G can be explored in t steps with k agents, then any temporal realization of G can be explored in $O((t+n)k\log n)$ steps with one agent.*

The next two lemmas show that taking subgraphs and edge contractions do not increase the arrival time of exploration in the worst case.

Lemma 2.2.3. [9] **(subgraphs)** *Let $G=(V,E)$ be a graph such that any temporal realization of G can be explored in t steps. Let $G'=(V',E')$ be a connected subgraph of G. Then any temporal realization of G' can also be explored in t steps.*

Lemma 2.2.4. [9] **(edge contraction)** *Let G be a graph such that any temporal realization of G can be explored in t steps. Let G' be a graph that is obtained from G by contracting edges. Then any temporal realization of G' can also be explored in t steps.*

Based on Lemmas 2.2.3 and 2.2.4, we can derive the following results:

Corollary 2.2.1. [9] **(minor)** *Let $G = (V, E)$ be a graph such that any temporal realization of G can be explored in t steps. Let $G' = (V', E')$ be a connected minor of G. Then any temporal realization of G' can also be explored in t steps.*

Corollary 2.2.2. [9] *Let $c < 1$ be a positive constant and $t(n)$ a function that is monotone increasing and satisfies $t(kn) = O(t(n))$ for any constant $k > 0$, e.g., a polynomial. Let $\mathcal{C}$ be a class of graphs such that any temporal realization of a graph G in the class can be explored in $t(n)$ steps, where n is the number of nodes of G. Let $\mathcal{D}$ be the class of graphs that contains all graphs that can be obtained from a graph G in $\mathcal{C}$ with n vertices by at most cn edge contractions. Then any temporal realization of a graph in $\mathcal{D}$ with n' vertices can be explored in $O(t(n'))$ steps.*

Now we consider how exploration schedules for the biconnected components of a graph can be combined into an exploration schedule for the whole graph. Recall that the block-cut tree (often also called the block graph) of a connected graph is a tree with a vertex for every block (biconnected component or bridge) and for every cut vertex of the graph, with an edge between a block and a cut vertex if the block contains that cut vertex [8]. If the vertices representing blocks in the block-cut tree of the graph have bounded degree, the next lemma shows that the total exploration time is on the order of the sum of the exploration times of the blocks.

Lemma 2.2.5. [9] *Assume that, for some function $t(n) \geq n - 1$, any temporal realization of any n-vertex graph from a class $\mathcal{C}$ of biconnected graphs can be explored in $t(n)$ steps. Let $G = (V, E)$ be a connected graph all of whose biconnected components belong to $\mathcal{C}$. Let $H_i = (V_i, E_i)$, for $1 \leq i \leq k$, be the blocks of G. If all vertices representing blocks in the block-cut tree of G have degree at most d, then any temporal realization of G can be explored in $O(d|V| + \sum_{i=1}^{k} t(|V_i|))$ steps.*

While static undirected connected graphs with n nodes can always be explored in less than $2n$ steps, the following lemma shows that there are temporal graphs that require $\Omega(n^2)$ steps.

Lemma 2.2.6. [9] *There is an infinite family of temporal graphs that, for every $r \geq 1$, contains a temporal graph $\mathcal{G}$ with $n = 2r$ vertices that requires $\Omega(n^2)$ steps to be explored.*

Corollary 2.2.3. [9] *For any number $k = o(n)$ of agents, there is an infinite family of temporal graphs such that each n-vertex temporal graph in the family cannot be explored in $o(n^2/k)$ steps.*

The underlying graph of the temporal graph constructed in the proof of Lemma 2.2.6 has maximum degree $|V| - 1$. For graphs with maximum degree bounded by d, we can show a lower bound of $\Omega(dn)$ in the following lemma.

Lemma 2.2.7. [9] *For every $d \geq 1$, there is an infinite family of temporal graphs with underlying graphs of maximum degree d that require $\Omega(dn)$ steps to be explored, where n is the number of vertices of the graph.*

For the proofs of the next three theorems, they use the fact that the Hamiltonian s-t path problem is **NP**-complete even if the input graphs are planar and have maximum degree 3 as shown by Garey, Johnson, and Tarjan [13].

Theorem 2.2.1. [9] *TEXP on planar graphs of maximum degree 3 is* **NP**-HARD.

They remark that temporal graphs whose underlying graph has maximum degree 2 are temporal realizations of paths or cycles. The exploration of temporal realizations of paths is trivial, as all edges of the path must exist in all steps of any temporal realization since we assume that the graph is connected in each step.

Theorem 2.2.2. [9] *Approximating TEXP with ratio $O(n^{1-\epsilon})$ is* **NP**-*hard for any constant $\epsilon > 0$.*

Theorem 2.2.3. [9] *For all ϵ, $\sigma > 0$, approximating TEXP with ratio $O(\Delta^{1-\epsilon})$ is NP-hard even if the underlying graphs have maximum degree at most $\Delta = \Omega((n^*)^\sigma)$ with n^* being the number of vertices of the temporal graph.*

Furthermore, they show that even the restriction to underlying graphs that are planar and have bounded degree is not sufficient to ensure the existence of an exploration schedule with a linear number of steps.

Theorem 2.2.4. [9] *Even if the underlying graph $G = (V, E)$ of a temporal graph $\mathcal{G}$ is planar with maximum degree 4 and the graph $\mathcal{G}_i$ in every step $i \geq 0$ is a simple path, an optimal exploration can take $\Omega(n \log n)$ steps, where $n = |V|$.*

Theorem 2.2.5. [9] *Any temporal graph whose underlying graph has treewidth at most k can be explored in $O(n^{1.5}k^{1.5} \log n)$ steps.*

2.3 Design Problems

So far, we have mainly presented problems in which a temporal graph is provided somehow (either in an offline or an online way) and the goal is to solve a problem on that graph. Another possibility is when one wants to design a desired temporal graph. In most cases, such a temporal graph cannot be arbitrary, but it has to satisfy some properties prescribed by the underlying application. This design problem was introduced and studied in [21]. An abstract definition of the problem is that we are given an underlying graph G and we are asked to assign labels to the edges of G so that the resulting temporal graph $\mathcal{G}$ minimizes some parameter while satisfying some connectivity property.

The parameters studied in [21] were the maximum number of labels of an edge, called the *temporality*, and the total number of labels, called the *temporal cost*. The connectivity properties of [21] had to do with the preservation of a subset of the paths of G in time-respecting versions. For example, we might want to preserve all reachabilities between nodes defined by G, in the sense that for every pair of nodes u, v such that there is a path from u to v in G there must be a temporal path from u to v in $\mathcal{G}$. Another such property is to guarantee in $\mathcal{G}$ time-respecting versions of all possible paths of G. All these can be thought of as trying to preserve a connectivity property of a static graph in the temporal dimension while trying to minimize some cost measure of the resulting temporal graph [22].

The provided graph G represents some given static specifications, for example the available roads between a set of cities or the available routes of buses in the city center. In scheduling problems it is very common to have such a static specification and to want to organize a temporal schedule on it, for example to specify the precise time at which a bus should pass from a particular bus stop while guaranteeing that every possible pair of stops are connected by a route. Furthermore, it is very common that any such solution should at the same time take into account some notion of cost. Minimizing cost parameters may be crucial as, in most real networks, making a connection available and maintaining its availability does not come for free. For example, in wireless sensor networks the cost of making edges available is directly related to the power consumption of keeping nodes awake, of broadcasting, of listening to the wireless channel, and of resolving the resulting communication collisions. The same holds for transportation networks where the goal is to achieve good connectivity

properties with as few transportation units as possible [22].

For instance, imagine that we are given a directed ring $u_1, u_2, \ldots, u_n$ and we want to assign labels to its edges so that the resulting temporal graph has a journey for every simple path of the ring and at the same time minimizes the maximum number of labels of an edge. In more technical terms, we want to determine or bound the *temporality* of the ring subject to the *all paths* property.

It is worth mentioning that the temporality (and the temporal cost) is defined as the minimum possible achievable value that satisfies the property, as, for example, is also the case for the chromatic number of a graph, which is defined as the minimum number of colors that can properly color a graph. Looking at Figure 3, it is immediate to observe that an increasing sequence of labels on the edges of path P_1 implies a decreasing pair of labels on edges (u_{n-1}, u_n) and (u_1, u_2). On the other hand, path P_2 uses first (u_{n-1}, u_n) and then (u_1, u_2) thus it requires an increasing pair of labels on these edges. It follows that in order to preserve both P_1 and P_2 we have to use a second label on at least one of these two edges; thus, the temporality is at least 2. Next, consider the labeling that assigns to each edge (u_i, u_{i+1}) the labels $\{i, n+i\}$, where $1 \leq i \leq n$ and $u_{n+1} = u_1$. It is not hard to see that this labeling preserves all simple paths of the ring. Since the maximum number of labels that it assigns to an edge is 2, we conclude that the temporality is also at most 2. Taking both bounds into account, we may conclude that the temporality of preserving all simple paths of a directed ring is 2.

Moreover, it holds that the temporality of graph G is lower bounded by the maximum temporality of its subgraphs, because if a labeling preserves all paths of G then it has to preserve all paths of any subgraph of G, paying every time the temporality of the subgraph. So, for example, if the input graph G contains a directed ring then the temporality of G must be at least 2 (and could be higher depending on the structure of the rest of the graph) [22].

Rings have very small temporality w.r.t. the all paths property; however, there is a large family of graphs with even smaller. This is the family of directed acyclic graphs (DAGs).

Definition 2.3.1. *A directed acyclic graph (DAG or dag) is a directed graph with no directed cycles. That is, it consists of vertices and edges, with each edge directed from one vertex to another, such that there is no way to start at any vertex v and follow a*

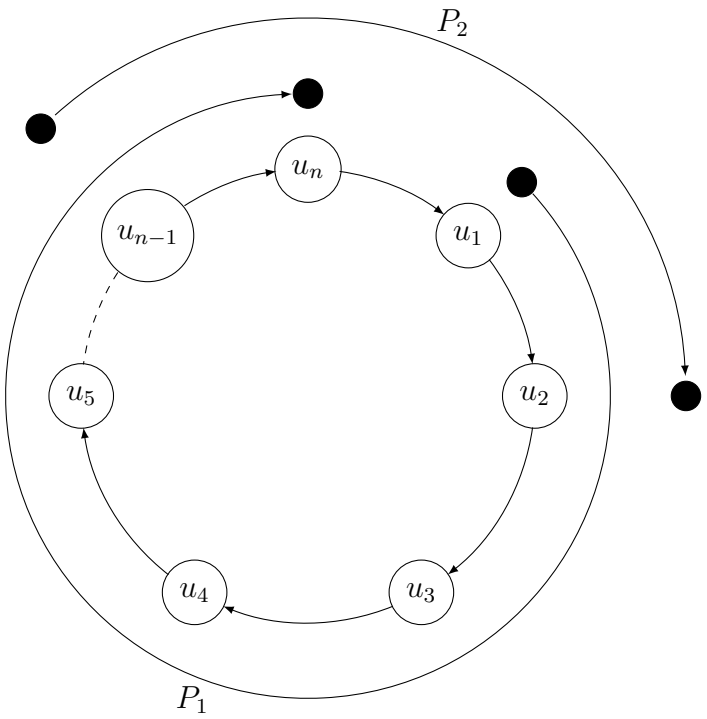

Figure 3: The temporal ring

consistently-directed sequence of edges that eventually loops back to v again.

DAGs have a very convenient property that they can be topologically sorted. In fact, DAGs are the only digraphs that satisfy this property [22].

In the examples above (Figure 3), all paths could be preserved by using very few labels per edge. One may immediately wonder whether converting all paths to journeys can always be achieved with few labels per edge, e.g., a constant number of labels. However, a more careful look at the previous examples may provide a first indication that this is not the case. In particular, the ring example suggests that cycles can cause an increase of temporality, compared to graphs without cycles, like DAGs. Of course, a single ring only provides a very elementary exposition of this phenomenon; however, as proved in [21], this core observation can be extended to give a quite general method for lower bounding the temporality. The idea is to identify a subset of the edges of G such that, for every possible permutation of these edges, G has a path following the direction of the permutation. Such subsets of edges, with many interleaved cycles, are called *edge-kernels* and it can be proved that the preservation of all paths of an edge-kernel on k edges yields a temporality of at least

k. To see this, consider an edge-kernel $K = \{e_1, e_2, \ldots, e_k\}$ and order increasingly the labels of each edge. Now take an edge with a maximum first label, move from it to an edge of maximum second label between the remaining edges, then move from this to an edge of maximum third label between the remaining edges, and so on. All these moves can be performed because K is an edge-kernel; thus, there is a path no matter which permutation of the edges we choose. As in step i we are on the edge e with maximum $i-th$ label, we cannot use the 1st, 2nd, ..., i-th labels of the next edge to continue the journey because none of these can be greater than the i-th label of e. So, we must necessarily use the $(i+1)$-th label of the next edge, which by induction shows that in order to go through the k-th edge in this particular permutation we need to use a k-th label on that edge [22].

2.4 Random Temporal Graphs

Another model of temporal graphs with succinct representation, is the model of random temporal graphs. Consider the case in which each edge (of an underlying clique) just picks independently and uniformly at random a *single time-label* from $[r] = \{1, 2, \ldots, r\}$. So it gets label $t \in [r]$ with probability $p = r^{-1}$

2.4.1 Temporal Path

We first calculate the probability that given a specific path $(u_1, u_2, \ldots, u_{k+1})$ of length k a journey appears on this path. We begin with the directed case. First, let us obtain a weak but elegant upper bound. Partition $[r]$ into $R_1 = \{1, \ldots, \lfloor r/2 \rfloor\}$ and $R_2 = \{\lfloor r/2 \rfloor + 1, \ldots, r\}$. Clearly, $P(journey) \leq P(\text{no } R_2R_1 \text{ occurs})$ as any journey assignment cannot have two consecutive selections s.t. the first one is from R_2 and the second from R_1. So, it suffices to calculate $P(\text{no } R_2R_1 \text{ occurs})$. Notice that the assignments in which no R_2R_1 occurs are of the form $(R_1)^i(R_2)^j$ for $i + j = k$, e.g. $R_1R_1R_2R_2R_2$ and there are $k+1$ of them. In contrast, all possible assignments are $2k$ corresponding to all possible ways to choose k times with repetition from $\{R_1, R_2\}$. So, $P(\text{no } R_2R_1 \text{ occurs}) = k/2^k$ (as all assignments are equiprobable, with probability $2-k$) and we conclude that $P(journey) \leq k/2^k$, which, interestingly, is independent of r; e.g. for $k = 6$ we get a probability of at most 0.09375 for a journey of length 6 to appear.

For any specific assignment of labels $t_1, t_2, \ldots, t_k$ of this path, where $t_i \in [r]([r] = \{1, 2, \ldots, r\})$, the probability that this specific assignment occurs in simply p^k. So all possible assignments have equal probability and we get

$$P(journey) = \frac{\#\text{ strictly increasing assignments}}{\#\text{ all possible assignments}} = \frac{\binom{r}{k}}{r^k}$$

Where $\binom{r}{k}$ follows from the fact that any strictly increasing assignment is just a unique selection of k labels from the r available and any such selection corresponds to a unique strictly increasing assignment. So, for example, for $k = 2$ and $r = 10$ we get a probability of $9/20$ which is a little smaller than $1/2$ as expected, due to the fact that there is an equal number of strictly increasing and strictly decreasing assignments but we also lose all remaining assignments which in this case are only the ties (that is, those for which $t_1 = t_2$).

Now it is easy to compute the expected number of journeys of length k. Let S be the set of all directed paths of length k and let Y_p be an indicator random variable which is 1 if a journey appears on a specific $p \in S$ and 0 otherwise. Let also X_k be a random variable giving the number of journeys of length k. Clearly, $E(X_k) = E(\sum_{p \in S} Y_p) = \sum_{p \in S} E(Y_p) = |S|$. P(a journey appears on a specific path of length k) $= n(n-1)\ldots(n-k)\binom{r}{k}r^{-k} \geq (n-k)^k\binom{r}{k}r^{-k}$. Now,if we set $n \geq r/\binom{r}{k}^{1/k} + k$, we get $E(X) \geq 1$. A simpler, but weaker, formula can be obtained by requiring $n \geq r + k$. In this case, we get $E(X) \geq \binom{r}{k}$. So, for example, a long journey of size $k = n/2$ that uses all available labels is expected to appear provided that $n \geq 2r$ (to see this, simply set $k = r$) [22].

2.4.2 Temporal Star Exploration

We now study the problem of star exploration in a temporal star graph on an underlying star graph $\mathcal{G}_s$ of n vertices, where the labels are assigned to the edges of $\mathcal{G}_s$ at random. In particular, each edge of $\mathcal{G}_s$ receives k labels independently of other edges, and each label is chosen uniformly at random and independently of others from the set of integers $\{1, 2, \ldots, \alpha\}$, for some $\alpha \in \mathbb{N}$. We call this a uniform random temporal star and denote it by $\mathcal{G}_s(\alpha, k)$. In this section, we investigate the probability of exploring all edges in a uniform random temporal star based on different values of α and k, thus partially characterizing uniform random temporal stars that can be fully explored or not, asymptotically almost surely.

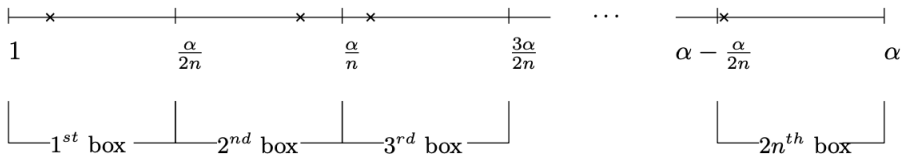

Figure 4: Splitting the time from 1 to α into $2n$ boxes to show the existence of at least one label per box, for every edge, asymptotically almost surely [1].

Theorem 2.4.1. [1] *If $\alpha \geq 2n$ and $k \geq 6n \ln n$, then the probability that we can explore all edges of $\mathcal{G}_s(\alpha, k)$ tends to 1 as n tends to infinity.*

Proof. [1] We consider the time-line from 1 to α and we split it into $2n$ consecutive equal-sized time-windows of size $\frac{\alpha}{2n}$ as shown in Figure 4. Let us henceforth refer to those as boxes and denote the i-th such box by B_i. The first box contains the labels $1, 2, \ldots, \frac{\alpha}{2n}$, the second box contains the labels $\frac{\alpha}{2n} + 1, \frac{\alpha}{2n} + 2, \ldots, \frac{\alpha}{n}$, and so on. See Figure 4.

We will show that for every edge of $\mathcal{G}_s$, there will be asymptotically almost surely at least one of its labels that falls in the first box, one of its labels that falls in the second box, etc. But first, let us note the following: □

Observation 2.4.1. [1] *If for every edge $e \in E$ and for every box B_i there is at least one label of e that lies within B_i, then there exists an exploration of $\mathcal{G}_s(\alpha, k)$.*

Proof. Assume that for every edge $e \in E$ and for every box B_i there is at least one label of e that lies within B_i. Fix an arbitrary order $e_1, e_2, \ldots, e_{n-1}$ of the edges of $\mathcal{G}_s$. Explore e_1 using its label that lies within B_1 to enter and its label that lies within B_2 to exit, explore e_2 using its label that lies within B_3 to enter and its label that lies within B_4 to exit, and so on and so forth. □

Note now that for a particular edge $e \in E$ and a particular box B_i of e, the probability that B_i contains none of the labels of e is:

$$Pr[B_i \text{ is empty}] = (1 - \frac{\frac{\alpha}{2n}}{\alpha})^k \leq (1 - \frac{\frac{\alpha}{2n}}{\alpha})^{6n \ln n} \leq \frac{1}{n^3},$$

So the probability that there is an empty box of e is:

$$Pr[\text{there is an empty } B_i \text{ of } e] \leq 2n.\frac{1}{n^3} = \frac{2}{n^2},$$

and so the probability that there exists an edge with an empty box is:

$$Pr[\text{there is an edge with an empty box}] \leq \#\text{edges}.\frac{2}{n^2} \leq \frac{2}{n}$$

Finally, the probability that we can explore all edges of $\mathcal{G}_s(\alpha, k)$ is:

$$Pr[\text{exploration}] \geq 1 - \frac{2}{n} \to 1, \text{ as } n \to +\infty$$

The latter completes the proof of the theorem.

Theorem 2.4.2. [1] *If $\alpha \geq 4$ and $k = 2$, then the probability that we can explore all edges of $\mathcal{G}_s(\alpha, k)$ tends to zero as n tends to infinity.*

Chapter 3

Contributions

In Section 3.1.1, we present known results from Erlebach et al. [9] regarding tight bounds on the exploration time of temporal cycles by a single agent from a worst-case start node. We also present our alternative proofs of similar (or same in the case of a lower bound) results with interesting new features. In Section 3.1.2 we present our new dynamic programming algorithm that for a given temporal cycle computes in polynomial time the optimal exploration time of a single agent starting at any node. In Section 3.1.3 we discuss implementation and empirical evaluation of the dynamic programming algorithm. Section 3.1 culminates with our proof of a conjecture of Erlebach et al. [9] that temporal cycles with constantly many chords can be explored in $O(n)$ time by a single agent from any start node. Along the way we introduce new tools for the analysis, such as a cycle cover, which might be of independent interest.

In Section 3.2 we study the exploration of $2 \times n$ temporal grids with various modifications. The starting point for the study is the algorithm of Erlebach et al. [9], which shows how to explore a standard $2 \times n$ temporal grid in $O(n \log^3 n)$ time. The original algorithm is rather non-intuitive with non-trivial recursion. We start by giving an overview of the Erlebach et al. algorithm, and we also present complete pseudocode for it (not present in [9]) to clarify the exposition. Then we adapt the algorithm to work in new settings of modified temporal grids.

3.1 Temporal Cycles

3.1.1 Exploration of Temporal Cycles: Tight Bounds

Erlebach et al. [9] showed that a single agent starting at an arbitrary vertex of a temporal cycle $\mathcal{C}$ of size n can explore it in at most $2n-2$ steps. Their proof is constructive and provides an algorithm for finding a temporal walk realising that exploration time. Moreover, they exhibited an instance of a temporal cycle $\mathcal{C}$ and a start node u such that a single agent starting at u requires at least $2n-3$ steps to explore $\mathcal{C}$. Thus, they established tight bounds on the exploration time of temporal cycles starting at a worst-case node. In this section, we present their results and proofs. We also present our alternative proofs of similar results, which involve different constructions and might be of independent interest. It is important to note that the upper bound results in this section establish *bounds* on the worst-case exploration time, and, in particular, they do not necessarily give an optimal exploration time for each temporal cycle and each starting point. Erlebach et al. do mention how to compute an optimal exploration time, but only at a high level. Our different algorithm to compute the optimal exploration time for each temporal cycle and each start node is based on dynamic programming and is presented and discussed in the sections following this one. We begin with the upper bound of Erlebach et al.

Theorem 3.1.1 (Erlebach et al. [9])**.** *Any temporal cycle $\mathcal{C}$ of length n can be explored in at most $2n-2$ steps by a single agent starting at any node. A schedule using this many steps can be computed in time linear in the total size of the graphs of the first $2n-2$ steps, i.e., in $O(n^2)$ time. If additionally an array $A : \{1,\dots,2n-2\} \to E$ is given that stores in $A[t]$ the edge that is missing in time step t, if any, then the running time can be improved to $O(n)$. Moreover, an optimal schedule for exploring a temporal cycle can be computed in polynomial time.*

Proof. Let $\mathcal{C}$ be a temporal cycle with n vertices labelled $v_1, v_2, \dots, v_n$ clockwise. Suppose, without loss of generality, that the agent a is initially located at v_1.

For now, disregard agent a and consider the temporal cycle at time $n-1$. Place n virtual agents $a_1, \dots, a_n$ on the vertices $v_1, \dots, v_n$ of the cycle, respectively, such that there is exactly one agent on each vertex. In each subsequent time step every virtual agent moves to its neighboring vertex clockwise, if possible. This means that the agent on v_i moves to v_{i+1} provided the edge (v_i, v_{i+1}) is present at that time. If

the edge is not present and as a result the agent cannot move clockwise, the agent will be eliminated and removed. As the temporal cycle $\mathcal{C}$ is connected at every step, there can be at most one edge missing at each time step. This implies that at each time step there can be at most one agent that cannot move to its neighboring vertex, and therefore will be eliminated. After $n-1$ steps we might lose at most $n-1$ agents. Thus there is one agent a_i that has moved forward for all the $n-1$ previous time steps. This agent has visited all nodes and therefore, has explored the cycle. So this virtual agent has explored the cycle by the time $2n-2$.

Now, we are ready to construct the exploration schedule of agent a. By Lemma 2.2.1, starting from vertex $v_1 \in v(\mathcal{C})$ at time 1 the agent a can move to vertex v_i in the first $n-1$ time steps. If the agent arrives at v_i in less than $n-1$ steps, the agent waits there until time $n-1$, at which point the agent simulates a_i by performing the clockwise exploration during the following $n-1$ time steps. Thus, the agent a explores the entire cycle by the time $2n-2$. We refer to the first part of this exploration (i.e., application of Lemma 2.2.1) as the first phase, and the second part as the second phase.

This schedule can be computed efficiently as follows. Consider the second phase and maintain the set of agents that have not yet disappeared. For each step $i = n, \ldots, 2n-2$, spend $O(1)$ time to determine the agent that starts $i-n+1$ vertices counterclockwise to the missing edge, i.e., determine the agent that disappears, if any. Finally, take one of the agents that remains in the set and compute a schedule for agent a to reach this virtual start vertex during the first phase. If we spend $O(n^2)$ time to iterate over the graphs of the first $2n$ steps and build the array A, then it is easy to see that the remaining computation can be done in $O(n)$ time.

An optimal exploration schedule can be computed in polynomial time as follows. By shortcutting backward and forward moves of the agents such that no vertices are skipped completely, any optimal schedule can be converted into one with the same arrival time that falls into one of these types: move clockwise around the cycle; move counter-clockwise around the cycle; move clockwise to some vertex v, then counter-clockwise until the cycle is explored; move counter-clockwise to some vertex w, then clockwise until the cycle is explored. The types can be enumerated in polynomial time, and the optimal schedule for each type can be calculated in a greedy way. The best of these schedules can then be output as the optimal exploration schedule for

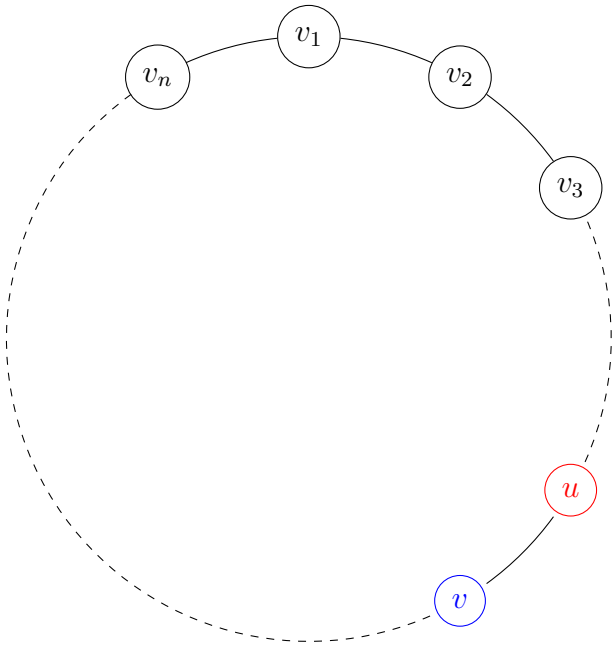

Figure 5: Meeting of two agents a_r and a_ℓ at time T in our alternative proof of the first part of Theorem 3.1.1.

the given temporal cycle. □

Next, we present our alternative proof of a slightly weaker upper bound of $2.5n$. Although our upper bound is slightly worse, it has some additional interesting features: we introduce only 4 virtual agents, and a good schedule is guaranteed to exist among the schedules corresponding to those 4 virtual agents.

Theorem 3.1.2. *Any connected temporal cycle $\mathcal{C}$ with n nodes can be explored in at most $2.5n$ time steps by a single agent starting at any node.*

Proof. As before, let $\mathcal{C}$ be a temporal cycle with n vertices labelled $v_1, \ldots, v_n$ clockwise. We construct (possibly partial) exploration schedules for four agents $a_r, a_\ell, a_{r'}, a_{\ell'}$, such that at least one of the agents is guaranteed to explore the entire cycle in at most $2.5n$ time steps.

We start the exploration from node v_1 using two agents a_r and a_ℓ. The agent a_r moves clockwise and the agent a_ℓ moves counterclockwise on the cycle. As $\mathcal{C}$ is connected, at each time step there can be at most one edge missing. We say two agents have met if they are at the two endpoints of an edge. See Figure 5.

Let $T > 1$ be the earliest time a_ℓ and a_r meet. It is easy to see that due to connectedness assumption prior to meeting at least one agent moves to a neighboring vertex in each time step (the same property holds after the agents pass each other), therefore $T \leq n$. Suppose at time T, a_r is at u and a_ℓ is at v.

If the edge $\{u, v\}$ is present at $T + 1$ then the agents continue exploring and return to v_1 within the next n steps. Thus, either agents' schedule can be used for the exploration of the entire cycle in at most $2n$ steps.

Otherwise (i.e., the edge $\{u, v\}$ is absent at $T + 1$) spawn two new agents at time T: $a_{r'}$ on u and $a_{\ell'}$ on v, and assume their walks were the same as those of a_r and a_ℓ, respectively, during the first T time steps. Then the behavior of agents diverges. The agents $a_{r'}$ and $a_{\ell'}$ start moving in the opposite directions.

Suppose that $\{u, v\}$ is missing for T' consecutive time steps starting at time $T+1$. Observe that during these T' steps while a_r and a_ℓ stay still (because of a missing edge $\{u, v\}$), both agents $a_{r'}$ and $a_{\ell'}$ move. Consider two cases.

Case $T' \geq n/2$: in this case agents $a_{r'}$ and $a_{\ell'}$ will pass each other, and at least one of them will finish the exploration after additional n time steps after the passing. The overall time taken by that agent is then $T + n/2 + n \leq n + n/2 + n = 2.5n$.

Case $T' < n/2$: in this case the original agents a_r and a_ℓ will pass each other at time $T+T'+1$, and at least one agent will finish the exploration in extra n steps after that. Thus, that agent's exploration time will be $T + T' + n \leq n + n/2 + n = 2.5n$.

□

Next we turn to the lower bound on the exploration time of a connected temporal cycle by a single agent from a worst-case start node. First, we present the original proof of Erlebach et al.

Theorem 3.1.3 (Erlebach et al. [9])**.** *For any $n \geq 3$, there is a temporal cycle of length n and a start node u, such that a single agent starting at u requires at least $2n - 3$ steps to explore the cycle starting from u.*

Proof. Assume that u, v, w are three subsequent vertices in this order of the cycle and the agent is initially at u. Let the edge $\{u, v\}$ be absent for the first $n - 2$ steps, and let the edge $\{v, w\}$ be absent in all steps after that. The agent cannot traverse the edge $\{v, w\}$ as it can reach neither v nor w before the edge disappears forever. So, the only two candidates for an optimal exploration schedule are the following: We

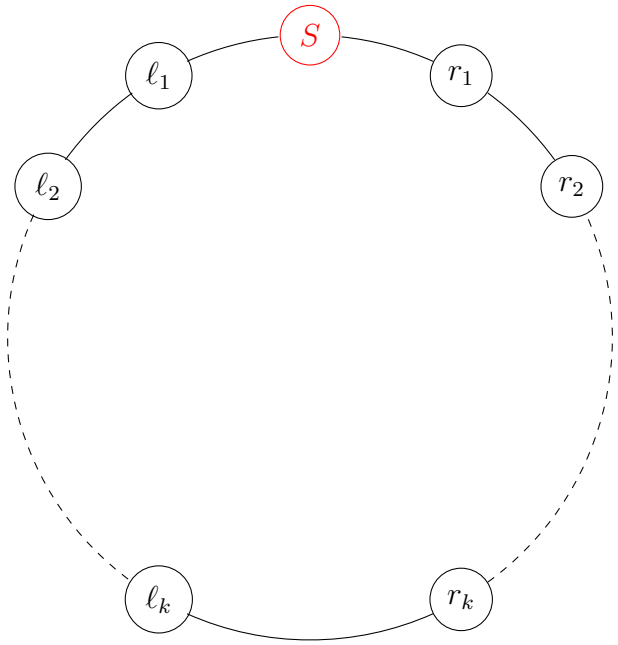

Figure 6: The underlying graph of a temporal cycle $\mathcal{C}$ in our alternative proof of Theorem 3.1.3

can either wait at u until $\{u, v\}$ is available ($n-2$ steps), move to v (1 step) and then walk to w ($n-1$ steps), giving a total of $2n-2$ steps, or walk to w in $n-2$ steps and then from w to v in $n-1$ steps, giving a total of $2n-3$ steps. □

Next, we present an alternative construction and an alternative proof of the same lower bound for odd values of n.

Our alternative proof of Theorem 3.1.3. We define the temporal cycle $\mathcal{C}$ as follows. Let the start node be S. There are k nodes labelled $r_1, r_2, \ldots, r_k$ in the clockwise order following S. There are k nodes labelled $\ell_1, \ell_2, \ldots, \ell_k$ in the counterclockwise order preceding S. See Figure 6.

In time step 1 the edge $\{S, \ell_1\}$ is removed. In the following time steps prior to $2k$, at each time step $2i$ the edge $\{r_i, r_{i+1}\}$ will be removed and at each time step $2i-1$ the edge $\{\ell_i, \ell_{i+1}\}$ will be removed. After time $2k-1$ the edge $\{\ell_k, r_k\}$ will be removed for all eternity. Consider the first $2k-1$ time steps, then we claim that:

1. The earliest time the agent starting at S can arrive at r_j is $2j-1$, and respectively the earliest arrival time the agent can arrive at ℓ_j is $2j$.

2. Both ℓ_k and r_k cannot be visited by $2k-1$.

This claim can be proved by induction on j. For $j=1$ it is clear that the agent can visit r_1 by $t=1$, while the earliest time the agent can arrive at ℓ_1 is $t=2$ (since the edge $\{S,\ell_1\}$ is missing in the first time step). Now let's assume the claim holds for $j \geq 1$. Suppose for contradiction that the agent managed to reach r_{j+1} at time $< 2(j+1)-1 = 2j+1$. The agent could not have reached r_{j+1} at time $2j$, since the edge from r_j to r_{j+1} is missing at that time. Also, the agent could not have taken the longer route around the circle (there is not enough time). Thus, the agent must have arrived at r_{j+1} at time $2j-1$. Meaning that the agent must have managed to arrive at r_j on or before $2j-2$. This contradicts the inductive assumption. The case of ℓ_j is handled similarly.

Thus, to explore the entire cycle starting from S the agent needs to visit either r_k or ℓ_k during the first $2k-1$ (or $2k$) steps, but since node ℓ_k (respectively, r_k) remains unvisited and edge $\{\ell_k, r_k\}$ is missing from that time onward, the agent must travel around the entire cycle to explore it completely. This results in the overall exploration time of $2k+2k-1 = 4k-1$. Observe that $n = 2k+1$, therefore the exploration time is at least $2n-3$.

□

3.1.2 Exploration of Temporal Cycles: Exact Dynamic Programming Algorithm

In this section we design a new dynamic programming algorithm to compute the optimal exploration time of a connected temporal cycle by a single agent starting from any node. In fact, our algorithm computes exploration times for all starting nodes simultaneously.

Before stating our algorithm, we need to extend the well known *Breadth First Search (BFS)* algorithm from the static setting to the temporal setting. The classical BFS algorithm in the static setting starts by visiting a node and then visits all its neighbors, then it visits all unvisited neighbors of those neighbors and so on. It could be visualized as sending out a wave out of the start node. The BFS produces shortest unweighted paths in the static setting. The BFS in the temporal setting is more involved and can be used to compute either foremost paths or shortest paths. We are

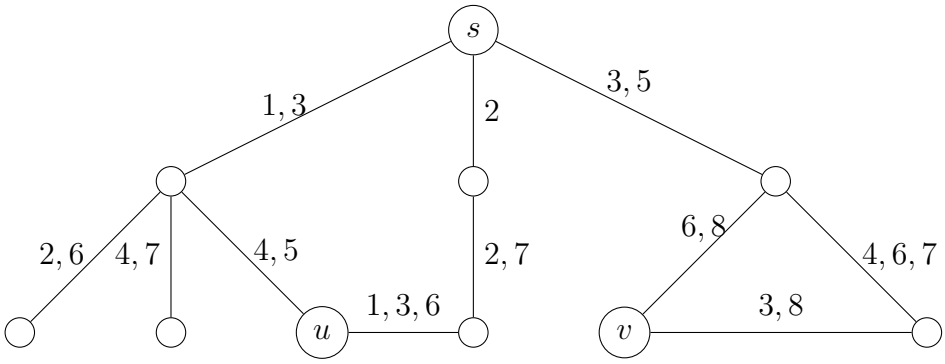

Figure 7: By running the TBFS algorithm on this graph starting at node s at time 0 we have: $tbfs[s][u][0] = 4$ and $tbfs[s][v][0] = 6$.

interested in the version of the temporal BFS that computes foremost paths starting at a particular node. See Huang et al. [15] for a slightly more general version of the TBFS that computes both foremost paths and shortest paths simultaneously. The TBFS, which we present in this section suffices for our purposes.

The goal of *Temporal Breadth First Search (TBFS)* algorithm is to populate the table $tbfs[u][v][t]$, where u and v are nodes of the given temporal graph $\mathcal{G}$ and t is a particular time step. The table is maintained as a global variable and the entry $tbfs[u][v][t]$ has the following meaning:

$$tbfs[u][v][t] = \text{the earliest arrival time at } v \text{ by an agent starting at node } u \text{ at time } t \text{ via a feasible temporal walk in } \mathcal{G}.$$

Algorithm 1 presents the pseudocode for the TBFS algorithm running from a start node s and start time $time$. By running TBFS from multiple start nodes and start times, one can pre-populate the entire $tbfs$ table. A single execution of the TBFS algorithm is akin to Dijkstra's algorithm. During the runtime $tbfs[s][v][time]$ maintains a monotonically decreasing upper bound on the earliest arrival time at v by an agent starting at s at time $time$. Initially, $tbfs[s][s][time] = time$ and $tbfs[s][v][time] = \infty$ for $v \neq s$ gives us tightest possible upper bounds given that we haven't processed anything from the graph yet. The algorithm maintains a min-priority queue of nodes with keys being $tbfs[s][v][time]$. When a node u is taken off the queue, it means that its entry $tbfs[s][u][time]$ has been determined correctly. Processing all outgoing

edges from u, we discover potentially new ways of reaching neighbors of u. Observe that for each edge $\{u, v\}$ it suffices to consider the corresponding time-edge $(\{u, v\}, t')$ with smallest t' subject to $t' > tbfs[s][u][time]$, and update $tbfs[s][v][time]$ if going through u results in an earlier arrival at v than what we knew before. Correctness can be shown by induction similar to the standard argument used for Dijkstra's algorithm.

Algorithm 1: Temporal Breadth First Search

Function `FillTBFS`$(\mathcal{G}, s, time)$**:**

- **for** $v \in V(\mathcal{G})$ **do**
  - $tbfs[s][v][time] \leftarrow \infty$;
- $tbfs[s][s][time] \leftarrow time$;
- $Q \leftarrow V(\mathcal{G})$ // a min-priority queue with the key of v being $tbfs[s][v][time]$;
- **while** $Q \neq \emptyset$ **do**
  - $u \leftarrow Q$.ExtractMin() ;
  - **for** $v \in adj[u]$ **do**
    - $E_{u,v} \leftarrow \{(u, v, t') \in E(\mathcal{G}) : t' > tbfs[s][u][time]\}$;
    - **if** $E_{u,v} = \emptyset$ **then**
      - **continue**;
    - $firstTime \leftarrow \min\{t' : (u, v, t') \in E_{u,v}\}$;
    - **if** $tbfs[s][v][time] > firstTime$ **then**
      - $tbfs[s][v][time] \leftarrow firstTime$;
      - Q.DecreaseKey(v, $tbfs[s][v][time]$);

Now, we are ready to describe our dynamic programming algorithm to compute optimal exploration time of connected temporal cycles. Let $\mathcal{C}$ be a given temporal cycle of size n and assume, without loss of generality, that the nodes are $\{1, 2, \ldots, n\}$ in clockwise order. For $i, j \in \{1, \ldots, n\}$ we define a (half-open) *interval* from i to j, denoted by $[i, j)$, to consist of those nodes that are encountered in clockwise order (with wrap-around) starting at node i and ending at a node prior to j. In other words, if $1 \leq i < j \leq n$ then $[i, j) = \{i, i + 1, \ldots, j - 1\}$ and if $1 \leq j < i \leq n$ then $[i, j) = \{i, i + 1, \ldots, n, 1, 2, \ldots, j - 1\}$. If $i = j$ then we define $[i, j) = \{1, 2, \ldots, n\}$. The algorithm populates the dynamic programming table $\mathcal{D}[i][j][k][t]$, where $1 \leq i \leq$

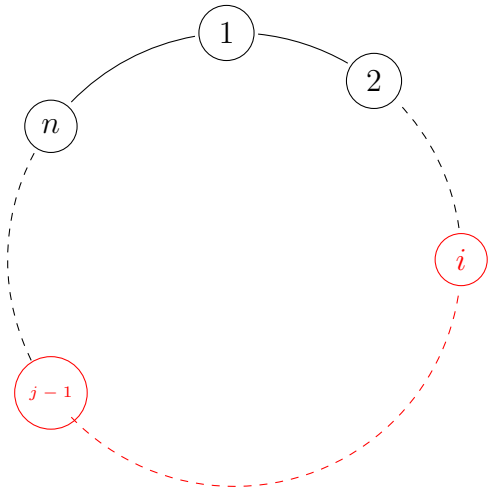

Figure 8: A temporal cycle in which the red nodes have been visited.

$n, 1 \leq j \leq n, k \in \{i, j-1\}, t \leq 2n-1$, where the bound on t follows from the results of Section 3.1.1. Thus, the table is of size $O(n^3)$. he table entry $\mathcal{D}[i][j][k][t]$ has the following intended meaning (see Figure 8):

$$\begin{aligned}\mathcal{D}[i][j][k][t] = &\text{ the smallest time needed to finish the exploration of } \mathcal{C} \\ &\text{by the agent located at node } k \text{ at time } t \text{ assuming that} \\ &\text{all the nodes in the interval } [i, j) \text{ have already been explored.}\end{aligned}$$

The base cases are given by $i = j \in \{1, 2, \ldots, n\}$. According the definition of the interval $[i, j]$ for $i = j$ and the above meaning of $\mathcal{D}[i][i][i][t]$, the agent has already explored the entire cycle, therefore we set $\mathcal{D}[i][i][i][t] = 0$ for all i and all t.

Assuming that the above table has been correctly computed, the smallest exploration time of an agent starting at node i at time 0 is stored in $\mathcal{D}[i][i+1][i][0]$. This is how the dynamic programming table can be used to answer the original question of exploration time starting from any node.

Next, we discuss how to compute the dynamic programming table. After prepopulating the entries corresponding to the base cases, the main idea is as follows: having visited all nodes in $[i, j)$ the agent located at $k \in \{i, j-1\}$ tries to extend the visited interval to either $[i, j+1)$ or to $[i-1, j)$. This requires the agent travelling from k to either $j+1$ or from k to $i-1$, respectively, and then finishing the exploration

of the rest of the cycle, which can be looked up in the previously computed entries. The time necessary for the agent to travel from k to either $j+1$ or $i-1$ can be computed using the temporal BFS. To fill in the entry corresponding to $[i, j)$ we just choose the better of these two options. Due to the topology of the cycle there is no other possibility of extending the explored region. More specifically, the table entries are computed as follows:

$$\mathcal{D}[i][j][j-1][t] = \min\{tbfs[j-1][j][t] + \mathcal{D}[i][j+1][j][tbfs[j-1][j][t]], \\ tbfs[j-1][i-1][t] + \mathcal{D}[i-1][j][i-1][tbfs[j-1][i-1][t]]\},$$

where index arithmetic is done with a wrap-around. Similarly, we have

$$\mathcal{D}[i][j][i][t] = \min\{tbfs[i][j][t] + \mathcal{D}[i][j+1][j][tbfs[i][j][t]], \\ tbfs[i][i-1][t] + \mathcal{D}[i-1][j][i-1][tbfs[i][i-1][t]]\}.$$

3.1.3 Exploration of Temporal Cycles: Empirical Results

Algorithm 2 shows the pseudocode for our dynamic programming algorithm computing optimal exploration time of a temporal cycle by a single agent from any start node, which was presented in Section 3.1.2. Note that the Temporal BFS algorithm must be run before the `Explore` algorithm in order to pre-populate the $tbfs$ table.

We implemented this algorithm in Java programming language and ran it on different temporal cycles inputs. Results are summarized in Tables 1 and 2. Observe that in every connected temporal cycle $\mathcal{C}$, there is always a node $v \in V(\mathcal{C})$, starting from which a single agent can visit all nodes in $n-1$ steps. This is easy to see and follows from the second phase of the schedule constructed in in Theorem 3.1.1.

Algorithm 2: Exploring a temporal cycle

$tbfs$ is a 3 dimensional array ;

D is a 4 dimensional array ;

for $v \in V$, $u \in V$, *and time* t **do**

  $tbfs[u][v][t] \leftarrow \infty$;

  $D[u][v][u][t] \leftarrow \infty$;

  $D[u][v][v][t] \leftarrow -1$;

for $s \in V$ *and time* t **do**

  `FillTBFS`$(\mathcal{C}, s, t)$;

for $v \in V$ **do**

  `Explore`$(v, v-1, v, 0)$;

Function `Explore`$(\mathcal{C}, i, j, k, t)$**:**

  if $D[i][j][k][t] \neq -1$ **then**

    return $D[i][j][k][t]$;

  $\mathcal{C} = \bigcup_{i=1}^{2n} \mathcal{C}_t$;

  $\forall t, V(\mathcal{C}_t) = \{v_1, \dots v_n\}$;

  $\forall t, \mathcal{C}_t$ is connected ;

  if $i = j - 1$ **then**

    return $tbfs[i][i][t]$

  if $k = i$ **then**

    return $D[i][j][k][t] \leftarrow$

    $min\{tbfs[i][j][t] + Explore(i, j, j, t + tbfs[i][j][t]), tbfs[i][i-1][t] +$

    $Explore(i-1, j-1, i-1, t + tbfs[i][i-1][t])\}$;

  else

    return $D[i][j][k][t] \leftarrow$

    $min\{tbfs[j-1][j][t] + Explore(i, j, j, t + tbfs[j-1][j][t]), tbfs[j -$

    $1][i-1][t] + Explore(i-1, j-1, i-1, t + tbfs[j-1][i-1][t])\}$;

Table 1 shows the result of running the algorithm on a set of temporal cycles with the number of vertices between 3 and 100 with a random missing edge in each time step. Table 2 shows the results of running the algorithm on a set of temporal cycles with a number of vertices between 3 and 100. In this set of temporal cycles, each cycle remains the same in every time step with exactly one specific missing edge in

all the steps. An obvious result of this algorithm can be generated by feeding static cycles to this algorithm. If we run this algorithm on a static cycle (the cycle remains unchanged during all time steps), the output will be $n-1$ from the best start node.

Interestingly, by looking at Tables 1 and 2, we can see that in many cases the exploration time for the worst starting vertex is about $\frac{3n}{2}$. Tables 3 and 4 show the result of running the algorithm on a data set of size 10, where each set contains 10 different temporal cycles. Table 3 shows the results of running the exploration algorithm on 10 sets of data. Each set has 10 different temporal cycles where each cycle has one missing edge by random. Similarly Table 4 shows the results of running the exploration algorithm on 10 sets of data. Each set has 10 different temporal cycles where each cycle has one specific missing edge in all steps.

3.1.4 Exploration of Temporal Cycles with Chords

Erlebach et al. proved an $O(n)$ upper bound on the exploration time of a temporal cycle with one chord in [9]. They have calculated the exploration time for such cycles by relying on the idea of Theorem 3.1.1.

Theorem 3.1.4 (Erlebach et al. [9]). *A connected temporal cycle of length n with one chord can be explored in $O(n)$ time by a single agent from any start node.*

Proof. Let the left and right cycle be the two cycles that contain the chord. Check how often the chord is present in the first $7n$ steps. If the chord is present in at least $5n$ steps, use $2n$ of these to explore the (left or right) cycle in which the start node is contained (which is possible by Theorem 3.1.1), n steps to move to the other cycle, and $2n$ steps to explore that cycle. Otherwise, there are at least $2n$ steps in which the chord is absent and the remaining graph is a cycle instance. The cycle can be explored in these steps. □

Erlebach et al. also conjectured that the same $O(n)$ bound must hold even for temporal cycles with constantly many chords.

Conjecture 3.1.1 (Erlebach et al. [9]). *A temporal cycle of length n with constantly many chords can be explored in $O(n)$ time by a single agent from any start node.*

In this section we prove this conjecture.

node	best start	worst start
51	50	59
3	2	3
62	61	65
42	41	44
19	18	20
63	62	66
85	84	87
17	16	17
24	23	27
5	4	6
45	44	49
68	67	71
11	10	13
91	90	94
74	73	80
55	54	56
39	38	43
22	21	22
35	34	37
27	26	29

Table 1: The result of the algorithm on the set of temporal cycles with a random missing edge. The best start shows the smallest time for the foremost path starting form a node and the worst start shows the biggest time for the foremost path starting from a node.

node	best start	worst start
67	66	99
76	75	112
68	67	100
33	32	48
57	56	84
41	40	60
54	53	79
47	46	69
65	64	96
24	23	34
26	25	37
23	22	33
30	29	43
14	13	19
4	3	4
15	14	21
88	87	130
13	12	18
9	8	12
96	95	142

Table 2: The result of the algorithm on the set of temporal cycles where cycles remain unchanged throughout the time. The best start shows the smallest time for the foremost path starting form a node and the worst start shows the biggest time for the foremost path starting from a node.

avg of # nodes	avg of time for best start	avg of time for worst start
42.2	41.2	44.4
53.7	52.7	61.2
49.3	48.3	51.9
42	41	45.4
49.7	48.7	53.8
51.2	50.2	56.7
39.6	38.6	44.7
57.5	56.5	62.2
44.8	43.8	49.3
54.6	53.6	59.8

Table 3: The result of the algorithm on 10 different set of 10 temporal cycles with a random missing edge.

avg # of nodes	avg of time for best start	avg of time for worst start
55	54	80.8
51.4	50.4	75.4
41.3	40.3	60.3
44.4	43.4	64.9
59.2	58.2	86.9
36.5	35.5	53.3
47.2	46.2	69.1
49.9	48.9	73.1
55.7	54.7	84.2
46.9	45.9	65.9

Table 4: The result of the algorithm on 10 different set of 10 temporal cycles with a specific starting point.

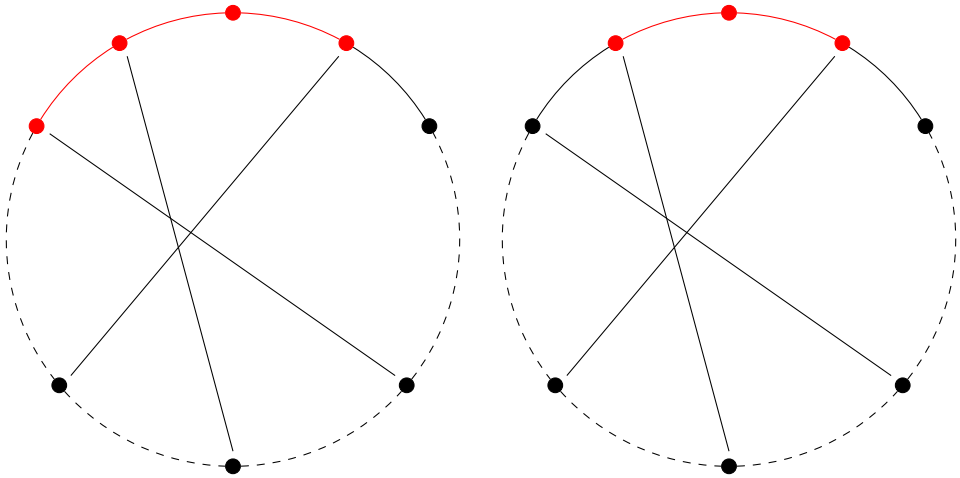

Figure 9: An arc in a cycle with chords on the left and a primary arc in a cycle with chords on the right

Theorem 3.1.5. *A connected temporal cycle $\mathcal{C}$ of length n with i chords can be explored in $O\left(((6i^2)(i!)(2e)^i)\, n\right)$ time by a single agent from any start node.*

We begin with a few definitions.

Definition 3.1.1. *An arc is a simple path on a cycle $\mathcal{C}$ between two nodes $u, v \in V(\mathcal{C})$ that does not use any chords of $\mathcal{C}$. We refer to u and v as endpoints of the arc.*

Definition 3.1.2. *A primary arc is an arc on which there is no vertex belonging to a chord, except the endpoints of the arc.*

An arc can be induced by a set of consecutive primary arcs. An example of an arc and a primary arc is shown in Figure 9.

High level idea of the proof of Theorem 3.1.5. The existence of an exploration walk in $O\left(((6i^2)(i!)(2e)^i)\, n\right)$ time steps is established by induction on i. Erlebach et al. have already handled the cases $i = 0$ (Theorem 3.1.1) and $i = 1$ (Theorem 3.1.4) for us. Now consider $i \geq 2$. The high-level idea is as follows. If one of the i chords is missing from sufficiently many time steps (to be specified later) then we can simply invoke the exploration walk for temporal cycles with $i - 1$ chords. This algorithm would be active during the time steps when the chord is missing and passive (not doing anything) during the other time steps. If such a chord does not exist then we must have a lot of time steps during which all i chords are present. Denote that set of time steps by T_G. Connectedness assumption and the existence of all i chords imply

some useful structural properties of the graph during T_G. Namely, every primary arc can have at most one edge missing from it. If we can cover the underlying graph by (potentially overlapping) cycles containing chords and arcs such that each cycle is connected during T_G, then we can explore these cycles one by one using the cycle-exploration algorithm. We call this cover a desirable k-cycle-cover if it contains k cycles. Thus, if a desireable k-cycle-cover exists with k depending only on i and independent on n then we are able to explore $\mathcal{C}$ in $O(n)$ (where the big-oh notation is hiding constants depending on i). The proof is finished by proving the existence of a desirable k-cycle-cover with k depending on i only. The rest of this section is dedicated to filling in the details of this argument.

Let T_i denote a bound on the exploration time of temporal cycles of size n with i chords by a single agent starting at a worst-case node. By Theorem 3.1.1 we have $T_0 \leq 2n$ and by Theorem 3.1.4 we have $T_1 \leq 7n$. We will show that the T_i for $i \geq 2$ satisfy the recurrence:

$$T_i \leq \binom{2i}{i-1} \times (6in - n) + iT_{i-1}.$$

First suppose that during the first T_i steps of a temporal cycle $\mathcal{C}$ with i chords, all i chords are present during at least $\binom{2i}{i-1} \times (6in - n)$ time steps. Let those time steps be denote T_G. Then we can explore the entire $\mathcal{C}$ during T_G. To that end, we define a new object that will guide the exploration – a desireable cycle cover. We begin with a slightly more general definition.

Definition 3.1.3. *A k-cycle-cover with respect to the temporal cycle $\mathcal{C}$ with i chords is a set of simple cycles $\{c_1, \ldots c_k\}$ with the following properties:*

1. *Each c_i is a simple cycle.*
2. *Each c_i contains at least one arc.*
3. *$\bigcup_{1 \leq i \leq k} c_i$ covers all the edges of $\mathcal{C}$.*

It follows that every c_i in a k-cycle cover has a structure of ≪arc, chord, arc, chord≫ or ≪arc, chord≫ in the underlying graph of $\mathcal{C}$. An example of a cycle cover is demonstrated in Figure 10. We sometimes refer to the c_i as sub-cycles.

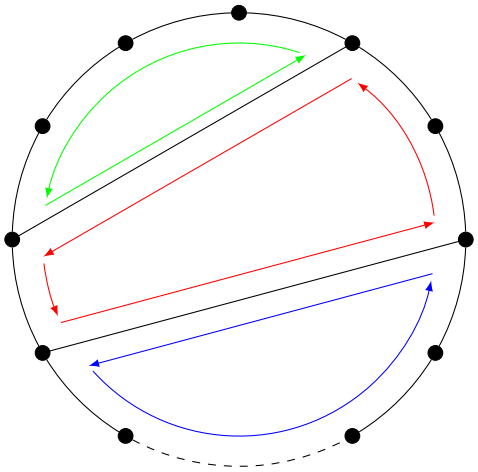

Figure 10: A *3-cycle-cover* in a cycle with two chords.

Definition 3.1.4. *A desirable k-cycle-cover $\mathcal{D}$ in a temporal cycle $\mathcal{C}$ with i chords during time steps S is a k-cycle-cover for which every cycle $c_i \in \mathcal{D}$ has at most one missing edge from the underlying graph of $\mathcal{C}$ during each time in S. This implies that each cycle in $\mathcal{D}$ is connected.*

Let $\mathcal{A} = \{A_1, \ldots, A_{2i}\}$ be the set of all primary arcs on $\mathcal{C}$ that has been induced by the i chords. Since $\mathcal{C}$ is connected at each time step and the underlying graph of $\mathcal{C}$ has i chords, it follows that at each time step there are at most $i+1$ edges missing from the graph. Also at most one edge can be missing from each primary arc without violating the connectedness assumption. Therefore, during each time $t \in T_G$ there can be at most one edge missing from some selection of $i+1$ primary arcs out of all $\mathcal{A}$ primary arcs. For simplicity, we assume that during each $t \in T_G$ exactly $i+1$ primary arcs are missing a single edge each. The set of such primary arcs at time t is called a *signature* of time t, denoted by $\sigma(t) \subseteq \mathcal{A}$. Observe that there there are $\binom{2i}{i+1} = \binom{2i}{i-1}$ different possible signatures. Since $|T_G| \geq \binom{2i}{i-1}(6in - n)$, by the pigeonhole principle, some signature must repeat at least $6in - n$ times during T_G. Let σ' denote such a signature and let T'_G denote the set of times such that for $t \in T'_G$ we have $\sigma(t) = \sigma'$. The following lemma demonstrates that we can construct a desirable $2i$-cycle-cover in $\mathcal{C}$ during time steps T'_G.

Lemma 3.1.1. *There is a desirable $2i$-cycle-cover in $\mathcal{C}$ during time steps T'_G.*

Proof. All the arguments below are performed with respect to time steps T'_G.

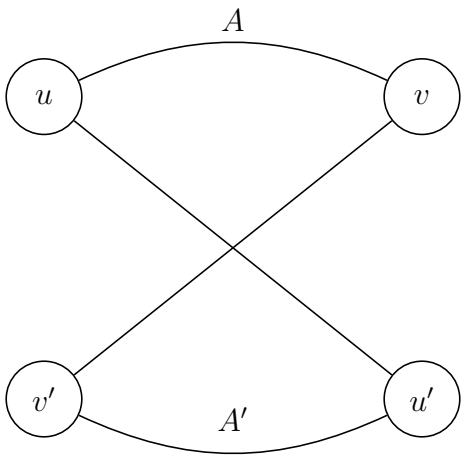

Figure 11: The two arcs induced by two chords on a cycle

Consider an arbitrary primary arc $A \in \mathcal{A}$. Let the two endpoint vertices of A be u and v and let the chords corresponding to them be $e_u = \{u, u'\}$ and $e_v = \{v, v'\}$. For simplicity, we assume that the chords do not share endpoints. The chords e_u and e_v divide the cycle $\mathcal{C}$ into four arcs. The arc between u and v, the arc between v and u', the arc between u' and v', the arc between v' and u. Let A' be the arc between u' and v'. See Figure 11. Now there are two cases to consider. Whether there is a missing edge on A' or not. It's important to note that there can be at most one missing edge on each primary arc as the graph $\mathcal{G}$ is connected at each step.

Case 1 There is no missing edge on arc A', i.e., primary arcs making up A' do not fall into the signature. In this case, the subcycle induced by two arcs A and A' and two chords e_u and e_v is a connected subcycle.

Case 2 There are missing edges on arc A'. As the graph $\mathcal{G}$ is connected at each time step, then there is a path (consisting of primary arcs and chords)from u' to v'. Let this path be P. Now the subcycle induced by the arc A, the path P and the two chords e_u and e_v, is a connected subcycle.

Since there are $2i$ primary arcs and we define one cycle per primary arc, this process gives a desirable $2i$-cycle-cover. □

We can use the desirable $2i$-cycle-cover to guide the exploration of $\mathcal{C}$ as follows: suppose the agent starts in some sub-cycle c_j of the cycle-cover. Then the agent uses at most $2n$ steps to explore c_j. Then it uses at most n steps to move to the next

unexplored sub-cycle in the cycle-cover and explores it in at most $2n$ more steps. This process is repeated until all sub-cycles in the cycle cover are explored guaranteeing that the entire $\mathcal{C}$ is explored. This takes at most $2i \times 2n + (2i-1) \times n = 6in - n$ time steps. Fortunately we have chosen $|T'_G| \geq 6in - n$, which means this exploration can be done during time steps T'_G.

Next consider the case when during the first T_i time steps, all i chords are present for fewer than $\binom{2i}{i-1}(6in-n)$ time steps. It means that at least one chord is missing from $\mathcal{C}$ for at least $T_i - \binom{2i}{i-1}(6in-n) \geq iT_{i-1}$ time steps. By the pigeonhole principle at least one fixed chord is missing for at least T_{i-1} time steps, which means that we can use the exploration procedure for cycles with $i-1$ chords to explore $\mathcal{C}$.

Next, we give a closed-form expression to bound T_i.

$$
\begin{aligned}
T_i &\leq \binom{2i}{i-1} \times (6in-n) + iT_{i-1} \\
T_{i-1} &\leq \binom{2(i-1)}{i-2} \times (6(i-1)n-n) + (i-1)T_{i-2} \\
&\vdots \\
T_2 &\leq \binom{4}{1} \times 11n + 2T_1 \\
T_1 &\leq 7n
\end{aligned}
$$

$$
T_i \leq \sum_{k=2}^{i} \binom{2k}{k-1}(6kn-n) + 7n
$$

Therefore we have:

$$
\begin{aligned}
T_i &\leq \binom{2i}{i-1}(6in-n) + i\binom{2(i-1)}{i-2}(6(i-1)n-n) + \cdots + i!\binom{4}{1}11n + i!(7n) \\
&\leq i!6in \sum_{j=1}^{i} \binom{2j}{j-1} \leq \left((6i^2)(i!)(2e)^i\right) n,
\end{aligned}
$$

where in the last step we used the bound $\binom{2j}{j-1} \leq \binom{2i}{i-1} \leq \left(\frac{2ie}{i-1}\right)^{i-1} \leq (2e)^i$.

Observe that $((6i^2)(i!)(2e)^i)$ is of $O(1)$ as long as i is a constant. Therefore we can conclude that exploration of a continuously connected temporal cycle with constantly many chords can be done in $O(n)$ time steps.

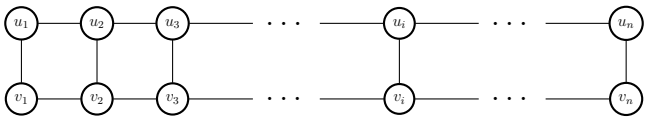

Figure 12: A $2 \times n$ grid which can be induced by the Cartesian product of P_2 and P_n.

Lastly, note that by Lemma 2.2.1, we can reach any node u from any node v in at most $n-1$ steps in $\mathcal{C}$. By repeated application of this observation we can explore $\mathcal{C}$ in at most n^2 steps. Thus, our bound provides an improvement over this trivial n^2 bound even when the number of chords is a slowly growing function of n. Our bound becomes meaningless (i.e., overtakes n^2) at around $O(\log n / \log \log n)$.

3.2 Temporal Grids

In this section we discuss the exploration of the temporal grid. A grid is the Cartesian product of two paths. For example, the Cartesian product of P_3 and P_7 creates a 3×7 grid.

Erlebach et al. [9] have discussed temporal $2 \times n$ grids and have presented an algorithm to explore such grids. In this section we still assume that graphs are continuously connected (which means that they are connected in every time step) and are simple. Moreover, we present an algorithm for the exploration of modified temporal grids.

3.2.1 Exploration of Temporal Grids

Definition 3.2.1. *A $2 \times n$ grid G consists of vertex set $V(G) = \{v_1, \ldots, v_n, u_1, \ldots, u_n\}$, where every two vertices v_i, v_{i+1} or u_i, u_{i+1} as well as v_i, u_i are adjacent. See Figure 12.*

Definition 3.2.2. *A modified grid G' with a pair of cross chord, is a grid for which there exist an i where $E(G') = E(G) \setminus \{(v_i, u_i), (v_{i+1}, u_{i+1})\} \bigcup \{(v_i, u_{i+1}), (v_{i+1}, u_i)\}$. See Figure 13.*

Theorem 3.2.1. [9] *Any temporal $2 \times n$ grid can be explored in $O(n \log n)$ steps with $4 \log n$ agents.*

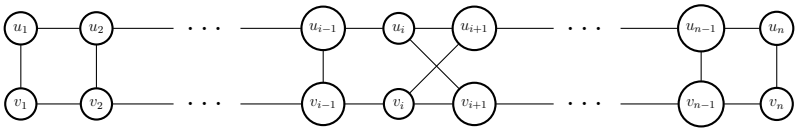

Figure 13: A $2 \times n$ modified grid.

Proof. This proof shows a more general statement. Given an underlying graph G' of $2 \times n$ gird and a subgrid G'' of size $2 \times n''$ of G' such that each pair of vertices in G'' is connected in G' in each step, then $4 \lg n'$ agents initially on some vertices of G'' can explore G'' in $T(n') = O(n' \lg n')$ time. The theorem follows by taking $G' = G'' = G$.

Each grid is explored by having its right half grid and left half grid explored separately. They start with exploring the left half H' of G'' . The idea is to move 4 agents to the corners of H', one to each corner, and all remaining $4(\lg n') - 4$ agents to a suitable middle location of H' using the first $2n'$ steps. For the next $T(n'/2) + n'/2$ time steps, in each time step where it is possible, they move the 2 agents l_1 and l_2 on the left corners of H' in parallel to the right using only horizontal edges. Similarly, they move the 2 agents r_1 and r_2 on the right corners to the left in parallel.

Let i and j be the number of moving steps of l_1 and r_1, respectively. The middle location is any position between the final position of l_1 and l_2 on the left and the final position of r_1 and r_2 on the right. If the agents on the left and on the right meet, they stop moving and H' is explored. In the same $T(n'/2) + n'/2$ steps where the 4 agents try to move, they explore recursively the subgrid H'' of H' consisting of the columns that are not visited by the 4 corner agents. In other words, at first the location of all the agents is calculated and then the exploring begins. every pair of l_i and l_{i+1} start exploring their subgrid. At each time step any pair of left agents that can move forward will explore the next vertices. The same thing happens for the right agents.

More precisely, there are at least $T(n'/2) + n'/2 - i - j \geq T(n'/2)$ steps in which neither the 2 agents l_1 and l_2 nor the 2 agents r_1 and r_2 move, and each pair of vertices of H'' is connected in H' in each of these steps. Therefore, the agents starting in the middle location can explore H'' in $T(n'/2)$ of those steps. Consequently, after the first $2n'$ steps to place the agents, the next $T(n'/2) + n'/2$ steps are enough to explore H. We subsequently explore the right half in the same way. The total time to explore G'' is $T(n') \leq 2(2n' + T(n'/2) + n'/2) = O(n' \lg n')$.

In other words, let $\mathcal{G} = \bigcup_{i=1}^{n^2} \mathcal{G}_t$ be the temporal grid with the vertex set $V(\mathcal{G}_t) =$

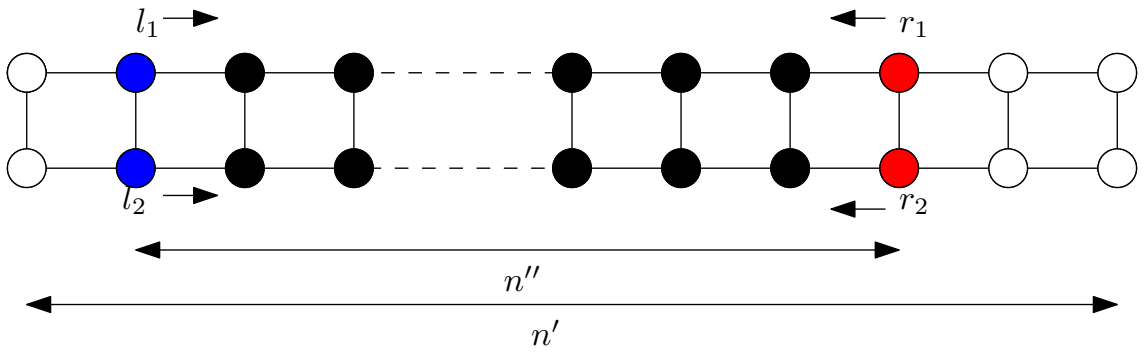

Figure 14: The situation as described in the proof of Theorem 3.2.1 A grid G' with a subgrid G'' (indicated by the black vertices) and the initial position of the agents to explore the left half of G.

$\{v_1, \dots v_n, u_1, \dots u_n\}$ and the set of agents S. The algorithm shows that, if it is given an underlying graph G being a grid of size $2 \times n$ and a subgrid G' of size $2 \times n'$ of G such that each pair of vertices in G' is connected in G in each step, then $4 \lg n$ agents initially on some vertices of G' can explore G'. Therefore the algorithm gets s and t as the beginning and the ending of the subgrid which is going to be explored. The explore procedure, examines whether there are any columns left to explore and then divide the subgrid into half and runs the move procedure on each half.

Move procedure calculates the number of columns in the grid it's going to explore, $n' = t - s + 1$, and place four agents on corresponding nodes on column s and t. The algorithm needs to separate the time steps in which the agents on v_s, u_s, u_t, v_t can move forward (good time steps) from those in which the four agents cannot move forward (bad time steps). Let T_g be the set of all good time steps and T_b be the set of bad time steps. The move procedures moves the four agents on v_s, u_s, u_t, v_t forward at $\forall t \in T_g$ until they are at $v_{s'}, u_{s'}, u_{t'}, v_{t'}$ and relocate the remaining agents in some suitable locations in columns $M = \{s' + 1, \dots, t' - 1\}$. Then it calls the Explore function recursively on the grid between columns s' and t' over time steps $t \in T_b$. The algorithm continues until four agents meet and there are no more columns left in the grid. Pseudocode 3 and 4 provide a high-level overview of the algorithm. □

Corollary 3.2.1. [9] *A temporal $2 \times n$ grid can be explored in $O(n \log^3 n)$ steps by one agent.*

Algorithm 3: Exploring a $2 \times n$ grid

$\mathcal{G} = \bigcup_{i=1}^{n^2} \mathcal{G}_t$, $\forall t, \mathcal{G}_t$ is connected ;

$\forall t, V(\mathcal{G}_t) = \{v_1, \dots v_n, u_1, \dots u_n\}$;

$f \leftarrow 1$;

$e \leftarrow n$;

$A \leftarrow \{A_1, \dots, A_{4 \log n}\}$;

$count \leftarrow 0$;

$time \leftarrow 0$;

Function `Explore`$(\mathcal{G}, f, e, s, t, S, A)$**:**

  S : set of agents, $|S| = 4 \log n$;

  if $s = t$ *or* $s + 1 = t$ **then**

    return $mid \leftarrow \lceil \frac{s+t}{2} \rceil$;

  `Move`$(\mathcal{G}, f, e, s, mid, S, A)$;

  `Move`$(\mathcal{G}, f, e, mid + 1, t, S, A)$;

3.2.2 Exploration of Modified Temporal Grids

Based on the previous section, here we present an algorithm for the exploration of modified grids. As stated before, a modified grid, is a grid in which two consequent vertical edges have been replaced by two crossed edges. Here we claim that the algorithm used in Theorem 3.2.1 can be used to explore such grids with some slight modifications.

Claim 3.2.1. *A temporal modified grid $\mathcal{G}'$ with a pair of chords is explorable in $O(n \log n)$ steps with $4 \log n$ agents.*

Proof. Let the set of vertices in $\mathcal{G}'$ be $\{u_1, \dots, u_n, v_1, \dots, v_n\}$. And let the crossed edges be (u_i, v_{i+1}) and (u_{i+1}, v_i). According to the algorithm presented in the proof of Theorem 3.2.1 we place 4 agents on the 4 corners of the left half of $\mathcal{G}'$. Let the agents be l_1, l_2, r_1, r_2. Agents l_1 and l_2 move simultaneously, and agents r_1 and r_2 move simultaneously as well. They only move if and only if both horizontal edges are present. When the 4 agents stop and can't move anymore, we repeat the algorithm recursively on the subgrid in between. The algorithm continues the same until either agents l_j, l_{j+1} reach v_i, u_i or agents r_k, r_{k+1} reach v_{i+1}, u_{i+1}. Without loss of generality,

Algorithm 4: Exploring a $2 \times n$ grid

Function `Move`$(\mathcal{G}, f, e, s, t, S, A)$**:**

$n' \leftarrow t - s + 1$;

Relocate 4 agents to v_s, v_t, u_s, u_t ;

$f \leftarrow s$;

$e \leftarrow t$;

$time \leftarrow time + 4n'$;

for $\mathcal{T} = time$ *to* $Cn' \log n' + time$ **do**

  $T_g \leftarrow \emptyset$;

  if $(v_s, v_{s+1}, \mathcal{T}) \in \mathcal{G}$ *and* $(u_s, u_{s+1}, \mathcal{T}) \in \mathcal{G}$ **then**

    $T_g \leftarrow T_g \cup \{\mathcal{T}\}$;

    $s \leftarrow s + 1$;

    $A_{count} \leftarrow A_{count+1} \leftarrow \mathcal{T}$;

  if $(v_{t-1}, v_t, \mathcal{T}) \in \mathcal{G}$ *and* $(u_{t-1}, u_t, \mathcal{T}) \in \mathcal{G}$ **then**

    $T_g \leftarrow T_g \cup \{\mathcal{T}\}$;

    $t \leftarrow t - 1$;

    $A_{count+2} \leftarrow A_{count+3} \leftarrow \mathcal{T}$;

  if $s = t$ *or* $s + 1 = t$ **then**

    return

$count \leftarrow count + 4$;

$M \leftarrow \{s + 1, \ldots, t - 1\}$;

$S_r \leftarrow S \backslash \{v_s, v_t, u_s, u_t\}$;

relocate S_r to some middle locations in M ;

$\mathcal{G}_\nabla \leftarrow \emptyset$;

for $t \in T_b$ **do**

  $\mathcal{G}_\nabla \leftarrow \mathcal{G}_\nabla \cup \mathcal{G}_t$;

`Explore`$(\mathcal{G}_\nabla, f, e, s + 1, t - 1, S_r, A)$;

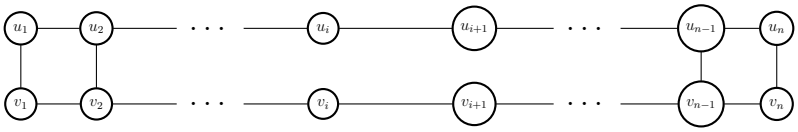

Figure 15: A temporal $2 \times n$ modified grid, in which the chords are missing and the horizontal edges of cross are present.

let us assume that agents l_j and l_{j+1} have reached vertices u_i and v_i respectively, and the size of the subgrid induced by the unvisited vertices is n'. Let $ch_1 = (u_i, v_{i+1})$, $ch_2 = (u_{i+1}, v_i)$, $e_1 = (u_i, u_{i+1})$ and $e_2 = (v_i, v_{i+1})$. There can be 4 different cases.

Case 1 If all 4 edges ch_1, ch_2, e_1 and e_2 are present, then the agents can move forward as they did.

Case 2 If 3 edges among $\{ch_1, ch_2, e_1, e_2\}$ are present. In this case either one chord is missing, or one of the horizontal edges is missing.

Case 2.1 If one of the chords is missing, then both horizontal edges are present. In this case both agents can move forward and continue with the algorithm.

Case 2.2 If one of the horizontal edges is missing (either e_1 or e_2), then both chords are present. In this case the agents can still move forward. Agent l_j goes over ch_1 and agent l_{j+1} goes over the edge ch_2. This means that the agents would move diagonally instead of horizontally. For the rest of the algorithm agent l_j would explore the lower set of vertices and agent l_{j+1} would explore the upper set of vertices.

Case 3 If only two edges among $\{ch_1, ch_2, e_1, e_2\}$ are present, there can be four different situations.

Case 3.1 If e_1 and e_2 are present then agents can move forward as they did. See Figure 15.

Case 3.2 If ch_1 and ch_2 are present then agents can move diagonally like Case 2.2 and move forward. See Figure 16.

Case 3.3 If one horizontal edge and one chord are present whose starting points is the same; then we can argue as follows. Due to symmetry, let us assume that e_1 and ch_1 are present. See Figure 17.

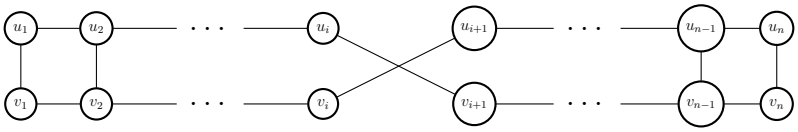

Figure 16: A temporal $2 \times n$ modified grid, in which the horizontal edges of cross are missing and the chords are present.

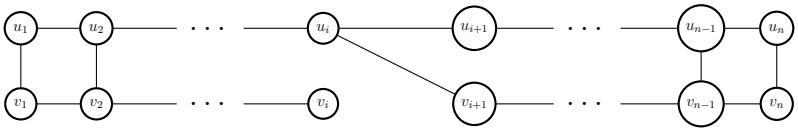

Figure 17: A temporal $2 \times n$ modified grid, in which one chord (ch_1) and one horizontal edge (e_1) are present.

In this case, we consider the next $\frac{n'}{2}$ steps. If both e_2 and ch_2 are missing, we can relocate 4 other agent and implement the algorithm recursively. We put both left agents on vertex u_i, and for the next step one agent goes over e_1 and one agent goes over ch_1. This is allowed as the unexplored vertices are connected in the outer subgrid.

Case 3.4 If one horizontal edge and one chord are present whose ending points are the same; then we can argue as follow. Due to symmetry, let us assume that e_1 and ch_2 are present. See Figure 18.

In this case the same reasoning as Case 3.3 works. If these two edges are present for $\frac{n'}{2}$ steps, then we repeat the algorithm recursively on the remaining vertices. The only difference in this case, is the relocation of the 4 new agents. We place the two left agents on vertices u_{i+1} and v_{i+1}. Then, the algorithm continues as it did.

Case 4 If only one edge is present then the algorithm follows the same procedure as

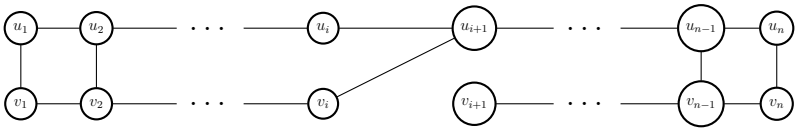

Figure 18: A temporal $2 \times n$ modified grid, in which one chord (ch_2) and one horizontal edge (e_1) are present.

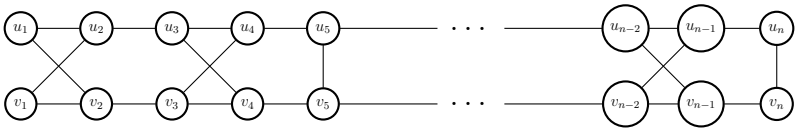

Figure 19: A $2 \times n$ temporal crossed grid.

in Theorem 3.2.1. If in the next $\frac{n'}{2}$ steps, only one edge is present then we can relocate the agents to the inner subgrid and repeat the algorithm.

Therefore, the same algorithm can be used for the temporal $2 \times n$ modified grid. □

Corollary 3.2.2. *A temporal $2 \times n$ modified grid can be explored in $O(n \log^3 n)$ steps by one agent.*

In the rest of this section we generalize Claim 3.2.1. In modified grids, there is only one pair of vertical edges that have been replaced by crossed edges. This means that our grid has only one pair of chords. We claim that a grid with more than one pair of chords, can be explored in $O(n \log^3 n)$ steps.

Definition 3.2.3. *A crossed grid G'' with m pairs of cross chords where $1 \leq m \leq \frac{n}{2}$, is a grid for which there is set $\{i_1, \ldots, i_m\}$ where for every $1 \leq k \leq m$, the set of edges is $E(G') = E(G) \setminus \{(v_{i_k}, u_{i_k}), (v_{i_k+1}, u_{i_k+1})\} \bigcup \{(v_{i_k}, u_{i_k+1}), (v_{i_k+1}, u_k)\}$. see Figure 19.*

Claim 3.2.2. *A temporal crossed grid $\mathcal{G}''$ with m pairs of chords is explorable in $O(n \log^3 n)$ steps by 1 agents.*

Proof. According to Claim 3.2.1, replacing two subsequent vertical edges with two crossed chords, does not affect the procedure of the algorithm. As in each step either the left agents or right agents can hit at most one pair of chords, the presence of other chords does not affect the algorithm. Therefore, we can follow the steps of the algorithm and explore the grid. □

Corollary 3.2.3. *A temporal $2 \times n$ crossed grid can be explored in $O(n \log^3 n)$ steps by one agent.*

Chapter 4

Conclusion

The study of temporal graphs is relatively young (even in such dynamic field as theoretical computer science), and we do not yet have intuition and a range of techniques comparable to what has been developed over many years for static graphs. Even seemingly simple tasks such as constructing temporal graphs (possibly with an underlying graph from a given family) that cannot be explored quickly is surprisingly difficult. Some of the methods that Erlebach et al. [9] presented and we used in this work to prove results for temporal graphs, e.g., the general conversion of multi-agent solutions to single-agent solutions, contribute to the formation of a growing toolbox for dealing with temporal graphs.

In this work, the problem of exploring two of the graph families have been studied. We presented an alternative dynamic programming algorithm regarding for the exploration of connected temporal cycles by a single agent from any start node, as well as gave alternative proofs of bounds on worst-case exploration time in this setting. Furthermore, we studied the exploration of temporal cycles with chords. Erlebach et al. have presented an algorithm to explore the temporal cycles with one chord in [9] and conjectured that it holds for temporal cycles with constant many chords. We proved their conjecture. The question of finding tight bounds for the exploration of temporal cycles with $m := m(n)$ chords for moderately quickly growing functions m remains unanswered and indicates an interesting future research direction.

Another family of graphs discussed in this thesis, is grids. We studied the exploration of $2 \times n$ temporal grids with some changes. We showed that the same algorithm used to explore a $2 \times n$ temporal grid, can be used to explore $2 \times n$ temporal crossed

grids. Finally, we proved that a $2 \times n$ temporal crossed grid is explorable in $O(n \log^3 n)$ steps.

However, there are numerous problems that are still unanswered regarding the exploration of temporal graphs. Our results directly suggest a number of questions for future work. In particular, deriving tight bounds on the largest number of steps required to explore a temporal graph whose underlying graph is an $m \times n$ grid, a bounded degree graph, or a planar graph would be interesting. It would also be interesting to study the approximability of TEXP for restricted underlying graphs, and to identify further cases of underlying graphs where the temporal exploration problem can be solved optimally in polynomial time. An interesting variation of TEXP is to allow the agent to make two moves (instead of one) in every time step.

Bibliography

[1] Eleni C Akrida, George B Mertzios, and Paul G Spirakis. The temporal explorer who returns to the base. In *International Conference on Algorithms and Complexity*, pages 13–24. Springer, 2019.

[2] Brenda Baker. Gossips and telephones. *Discrete Mathematics*, 2:191–193, 1972.

[3] Kenneth A Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks: An International Journal*, 28(3):125–134, 1996.

[4] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In *International Conference on Ad-Hoc Networks and Wireless*, pages 346–359. Springer, 2011.

[5] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[6] Augustin Chaintreau, Abderrahmen Mtibaa, Laurent Massoulie, and Christophe Diot. The diameter of opportunistic mobile networks. In *Proceedings of the 2007 ACM CoNEXT conference*, pages 1–12, 2007.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[8] Reinhard Diestel. *Graph theory: Springer graduate text gtm 173*, volume 173. Reinhard Diestel, 2012.

[9] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *International Colloquium on Automata, Languages, and Programming*, pages 444–455, 2015.

[10] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[11] Afonso Ferreira. Building a reference combinatorial model for manets. *IEEE network*, 18(5):24–29, 2004.

[12] Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *International Symposium on Algorithms and Computation*, pages 534–543. Springer, 2009.

[13] Michael R Garey, David S. Johnson, and R Endre Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[14] Petter Holme. Network reachability of real-world contact sequences. *Physical Review E*, 71(4):046119, 2005.

[15] Silu Huang, James Cheng, and Huanhuan Wu. Temporal graph traversals: Definitions, algorithms, and applications. *CoRR*, abs/1401.1919, 2014.

[16] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

[17] Gueorgi Kossinets, Jon Kleinberg, and Duncan Watts. The structure of information pathways in a social communication network. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 435–443, 2008.

[18] Vassilis Kostakos. Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023, 2009.

[19] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522, 2010.

[20] Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *ACM SIGACT News*, 42(1):82–96, 2011.

[21] George B Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 657–668. Springer, 2013.

[22] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[23] Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. *Journal of Parallel and Distributed Computing*, 74(1):2016–2026, 2014.

[24] Regina O'Dell and Rogert Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 104–110, 2005.

[25] Ramamoorthi Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 202–213. IEEE, 1994.

[26] Rob Shields. Cultural topology: The seven bridges of königsburg, 1736. *Theory, Culture & Society*, 29(4-5):43–57, 2012.

[27] John Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *ACM SIGCOMM Computer Communication Review*, 40(1):118–124, 2010.

[28] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996.

[29] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9):721–732, 2014.

[30] B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.