Thejaswi, Suhas; Gionis, Aristides

Pattern detection in large temporal graphs using algebraic fingerprints

Published in:
Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020

DOI:
10.1137/1.9781611976236.5

Published: 01/01/2020

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Thejaswi, S., & Gionis, A. (2020). Pattern detection in large temporal graphs using algebraic fingerprints. In C. Demeniconi, & N. Chawla (Eds.), *Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020* (pp. 37-45). (Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611976236.5

Pattern detection in large temporal graphs using algebraic fingerprints*

Suhas Thejaswi† Aristides Gionis‡

Abstract

In this paper, we study a family of pattern-detection problems in *vertex-colored temporal* graphs. In particular, given a vertex-colored temporal graph and a multi-set of colors as a query, we search for *temporal paths* in the graph that contain the colors specified in the query. These types of problems have several interesting applications, for example, recommending tours for tourists, or searching for abnormal behavior in a network of financial transactions.

For the family of pattern-detection problems we define, we establish complexity results and design an algebraic-algorithmic framework based on *constrained multilinear sieving*. We demonstrate that our solution can scale to massive graphs with up to hundred million edges, despite the problems being **NP**-hard. Our implementation, which is publicly available, exhibits practical edge-linear scalability and highly optimized. For example, in a real-world graph dataset with more than six million edges and a multi-set query with ten colors, we can extract an optimal solution in less than eight minutes on a haswell desktop with four cores.

1 Introduction

Pattern mining in graphs has become increasingly popular due to applications in analyzing and understanding structural properties of data originating from information networks, social networks, transportation networks, and many more. At the same time, real-world data are inherently complex. To accurately represent the heterogeneous and dynamic nature of real-world graphs, we need to enrich the basic graph model with additional features. Thus, researchers have considered *labeled graphs* [40], where vertices and/or edges are associated with additional information represented with labels, and *temporal graphs* [20], where edges are associated with timestamps that indicate when interactions between pairs of vertices took place.

In this paper we study a family of pattern-detection problems in graphs that are both *labeled* and *temporal*. In particular, we consider graphs in which each vertex is associated with one (or more) labels, to which we refer as *colors*, and each edge is associated with a timestamp. We then consider a *motif query*, which is a multi-set of colors. The problem we consider is to decide whether there exists a *temporal path* whose vertices contain exactly the colors specified in the motif query. A temporal path in a temporal graph refers to a path in which the timestamps of consecutive edges are strictly increasing. If such a path exists, we also want to find it and return it as output.

The family of problems we consider have several interesting applications. One application is in the domain of tour recommendations [11], for travelers or tourists in a city. In this case, vertices correspond to locations. The colors associated with each location represent different activities that can be enjoyed in that particular location. For example, activity types may include items such as museums, archaeological sites, restaurants, etc. Edges correspond to transportation links between different locations, and each transportation link is associated with a timestamp indicating departure time and duration. Furthermore, for each location we may have information about the amount of time recommended to spent in that location, e.g., minimum amount of time required to finish a meal or appreciate a museum. Finally, the multi-set of colors specified in the motif query represents the multi-set of activities that a user is interested in enjoying. In the tour-recommendation problem we would like to find a temporal path, from a starting location to a destination, which satisfies temporal constraints (e.g., feasible transportation links, visit times, and total duration) as well as the activity requirements of the user, i.e., what kind of places they want to visit.

Another application is in the domain of analyzing networks of financial transactions. Here, the vertices represent financial entities, the vertex colors represent features of the entities, and the temporal edges represent financial transactions between entities, annotated with the time of the transaction, amount, and possibly other features. An analyst may be interested in

*This research was supported by the Academy of Finland project "Adaptive and Intelligent Data (AIDA)" (317085), the EC H2020 RIA project "SoBigData++" (871042), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. We acknowledge the use of computational resources funded by the project "Science-IT" at Aalto University, Finland.

†Department of Computer Science, Aalto University, Finland.

‡Department of Computer Science, KTH Royal Institute of Technology, Sweden, and Department of Computer Science, Aalto University, Finland.

finding long chains of transactions among entities that have certain characteristics, for example, searching for money-laundering activities may require querying for paths that involve public figures, companies with certain types of contracts, and banks in off-shore locations.

In this paper we study the following problems:

k-TempPath: find a temporal path whose length is at least $k-1$;

PathMotif: find a temporal path whose vertices contain the set of colors specified by a motif query;

RainbowPath: find a temporal path of length at least $k-1$, whose vertices have distinct colors.

All the problems we consider are **NP**-hard; thus, there is no known efficient algorithm to find exact solutions. In such cases most algorithmic solutions resort to approximation schemes. In this paper we present an (exact) *algebraic approach* based on *constrained multilinear sieving* for pattern detection in temporal graphs.

The algorithms based on constrained multilinear detection offer the theoretically best-known results for a set of combinatorial problems including k-path [4], Hamiltonian path [5], graph motifs [6] and many more. The implementations based on multilinear sieving are known to saturate the empirical arithmetic and memory bandwidth on modern CPU and GPU micro-architectures. Furthermore, these implementations can scale to large graphs as well as large query sizes [7, 22].

Even though these algebraic techniques have been studied extensively in the algorithms community, they are not been applied to data-mining problems. To the best of our knowledge this is the first paper that applies these approaches for data mining and exploratory graph analysis. Furthermore, this is the first work that applies these techniques for pattern detection in temporal graphs.

Our key contributions are as follows:

- We introduce a set of pattern-detection problems that originate in the *vertex-colored* and *temporal* graphs. For the problems we define we present **NP**-hardness results, while showing that they are *fixed-parameter tractable* [10], meaning that, if we restrict the size of motif query the problems are solvable in polynomial time in the size of host graph.
- We present a general algebraic-algorithmic framework based on constrained multilinear sieving. Our solution exhibits edge-linear scalability. The applications of the algorithmic approach described in this work is not limited to temporal paths, but rather it can be extended to study information cascades, temporal arborescences and temporal subgraphs.
- We engineer an implementation of the algebraic algorithm and demonstrate that the implementation can scale for graphs with up to hundred million edges.
- Open-source release: our implementation and datasets are released as open source [33].

Due to space constraints, the proofs of our technical contributions are omitted from the conference proceedings. A more detailed version of this work is available [34].

2 Related work

Pattern detection and pattern counting are fundamental problems in data mining. In the context of paths and trees, pattern matching problems have been extensively studied in non-temporal graphs both in theory [14, 17, 29] as well as applications [3, 21]. For many restricted variants of path problems Kowalik and Lauri presented complexity results and deterministic algorithms with probably optimal runtime bounds [29]. Most of these problems are known to be fixed-parameter tractable and the best known randomized algorithms for a subset of path and subgraph pattern detection problems is due to Björklund et al. [4, 6]. Color coding can be used to approximately count the patterns in $\mathcal{O}^*(2^k)$ time, however, these algorithms require $\mathcal{O}^*(2^k)$ memory [1].[1] A practical implementation of color coding using adaptive sampling and succint encoding was demonstrated by Bressan et al. [8] for the pattern-counting problem. However, the techniques based on color coding are mostly used to detect and count patterns in graphs with no vertex labels.

Algebraic algorithms based on multilinear and constrained multilinear sieving are due to the pioneering work of Koutis [23–25], Williams [37], Koutis and Williams [26, 27]. The approach has been extended to various combinatorial problems using a multivariate variant of the sieve by Björklund et al. [4]. Dell et al. [12] used the decision oracles introduced by Björklund et al. to approximately count the motifs. A practical implementation of multilinear sieving and its scalability to large graphs has been demonstrated by Björklund et al. [7]. Furthermore, its parallelizability to vector-parallel architectures and scalability to large multi-set sizes was shown by Kaski et al. [22].

In the recent years there has been a lot of progress with respect to mining temporal graphs. The most relevant work includes methods for efficient computation of network measures, such as centrality, connectivity, density, motifs, etc. [20, 30], as well as mining frequent subgraphs in temporal networks [32, 36]. Path problems in temporal graphs are well studied [16, 38]. Many

[1] The $*$ in the notation $\mathcal{O}^*$ hides the polynomial factor.

variants of the path problems are known to be solvable in polynomial-time [38, 39]. Surprisingly, a simple variant to check the existence of a temporal path with waiting time constraints was shown to be **NP**-complete by Casteigts et al. [9], more strongly, they proved that the problem is **W**[1]-hard. A known variant of the temporal-path problem is finding top-k shortest paths. In this setting, one asks to find not only a shortest path, but also the next $k-1$ shortest paths, which may be longer than the shortest path [19]. Here by shortest path we mean that the total elapsed time of the temporal path is minimized. Note that the top-k shortest path is different from the k-TempPath problem studied in our work.

With the availability of social media data in the recent years there has been growing interest to study pattern mining problems in temporal graphs. Paranjape et al. [32] presented efficient algorithms for counting small temporal patterns. Liu et al. [31] presented complexity results and approximation methods for counting patterns in temporal graphs. However, they mainly study temporal graphs with no vertex-labels (colors). Kovanen et al. [28] studied a general variant of the temporal subgraph problem in temporal graphs with vertex labels. Aslay et al. [2] presented methodologies for counting frequent patterns with vertex and edge labels in streaming graphs. However, most of these approaches are limited to small pattern sizes (up to 3 vertices).

To the best of our knowledge, there is no existing work related to detecting and extracting temporal patterns with vertex labels. The problems considered in this paper are closely related to variants of classical problems such as orienteering problem, TSP and Hamiltonian path [15, 35]. A motivating application for the problems can be traced to the context of tour recommendations [11, 18].

3 Method overview

Our method relies on the *algebraic-fingerprinting* technique [26, 37]. As this technique is not well known in the data-mining community, we provide a bird's eye view. The approach is described in more detail in Section 6.

In a nutshell, the problem is to decide the existence of a pattern, or a structure in the data. The idea is to encode the pattern-discovery problem as a polynomial over a set of variables. The variables represent entities of the problem instance (e.g., vertices or edges), and their values represent possible solutions (e.g., whether a vertex belongs to a path). The challenge is to find a polynomial encoding that has the property that a solution to our problem exists if and only if the polynomial evaluates to a non-zero term. We can then verify the existence of a solution, using *polynomial identity testing*, in particular, by evaluating random substitutions of variables: if one of them does not evaluate to zero, then the polynomial is not identically zero. Thus, the method can give false negatives, but the error probability can be brought arbitrarily close to zero.

It should also be noted that an explicit representation of the polynomial can be exponentially large. However, we do not need to represent the polynomial explicitly, since we only need to be able to evaluate the variable substitutions very fast.

4 Terminology

In this section we introduce the basic terminology used in the paper.

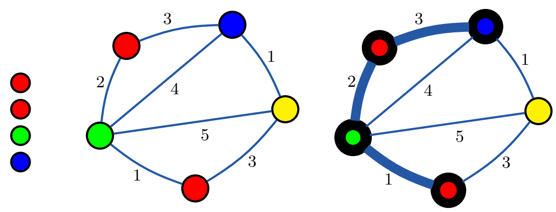
A *graph* G is a tuple (V, E) where V is a set of *vertices* and E is a set of unordered pairs of vertices called *edges*. We denote the number of vertices $|V| = n$ and the number of edges $|E| = m$. Vertices u and v are *adjacent* if there exists an edge $(u, v) \in E$. The set of vertices adjacent to vertex u is denoted by $N(u)$. A *walk* between any two vertices is an alternating sequence of vertices and edges $u_1 e_1 u_2 \ldots e_k u_{k+1}$ such that there exists an edge $e_i = (u_i, u_{i+1}) \in E$ for each $i \in [k]$.[2] We call the vertices u_1 and u_{k+1} the *start* and *end* vertices of the walk, respectively. The *length* of a walk is the number of edges in the walk. A *path* is a walk with no repetition of vertices.

A *temporal graph* G^τ is a tuple (V, E^τ), where V is a set of vertices and E^τ is a set of temporal edges. A *temporal edge* is a triple (u, v, j) where $u, v \in V$ and $j \in Z_{\geq 0}$ is a *timestamp*. The *maximum timestamp* in G^τ is denoted by t. The total number of edges at time instance $j \in [t]$ is denoted by m_j and the total number of edges in a temporal graph is $m = \sum_{j \in [t]} m_j$. A vertex u is adjacent to vertex v at timestamp j if there exists an edge $(u, v, j) \in E^\tau$. The set of vertices adjacent to vertex u at time step j is denoted by $N_j(u)$. The set of vertices adjacent to vertex u is denoted by $N(u) = \bigcup_{j \in [t]} N_j(u)$.

A *temporal walk* W^τ between any two vertices is an alternating sequence of vertices and temporal edges $u_1 e_1 u_2 e_2 \ldots e_k u_{k+1}$ such that there exists an edge $e_i = (u_i, u_{i+1}, j) \in E^\tau$ for all $i \in [k]$ and for any two edges $e_i = (u_i, u_{i+1}, j)$, $e_{i+1} = (u_{i+1}, u_{i+2}, j')$ it is $j < j'$. In other words, the timestamps of the edges should always be in strictly increasing order. The *length* of a temporal walk is the number of edges in the temporal walk. A *temporal path* is a temporal walk with no repetition of vertices.

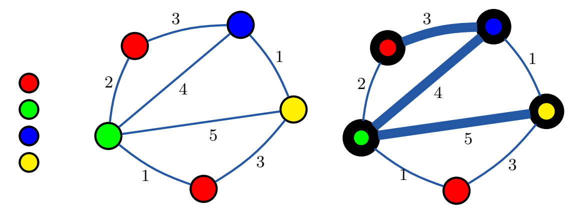
In the next section we will introduce a set of path problems in temporal graphs and an exact algorithm

[2] For convenience we represent $\{1, 2, \ldots, k\}$ as $[k]$.



(a) A motif query and a temporal graph (b) A PathMotif

Figure 1: An example of the PathMotif problem in temporal graphs.



(a) A motif query and a temporal graph (b) A RainbowPath

Figure 2: An example of the RainbowPath problem in temporal graphs.

based on *multilinear sieving* is presented in Section 6.

5 Path problems in temporal graphs

Let us begin our discussion with the k-path problem for static graphs before continuing to temporal graphs.

The k-path problem in static graphs. Given a graph $G = (V, E)$ and an integer $k \leq n$ the k-Path problem asks to decide whether there exists a path of length at least $k-1$ in G. The k-Path problem is **NP**-complete [15, ND29]. Fortunately, the problem is *fixed-parameter tractable*. The best known fixed-parameter tractable algorithm is due to Björklund et al. [4] and has complexity $\mathcal{O}^*(1.66^k)$.

The k-path problem in temporal graphs. Given a temporal graph $G^\tau = (V, E^\tau)$ and an integer $k \leq n$ the k-TempPath problem asks to decide whether there exists a *temporal path* of length at least $k-1$ in G^τ. For the hardness, a reduction from k-Path to k-TempPath is straightforward.

Lemma 5.1. *Problem k-TempPath is* **NP***-complete.*

Path motif problem in temporal graphs. Given a vertex-colored temporal graph $G^\tau = (V, E^\tau)$ and multi-set M of colors the PathMotif problem asks to decide whether there exists a temporal path P^τ in G^τ such that the vertex colors of P^τ agrees with M. An example of PathMotif problem is illustrated in Figure 1.

The PathMotif problem is **NP**-complete — a reduction from k-TempPath is straightforward.

Lemma 5.2. *Problem PathMotif is* **NP***-complete.*

Rainbow path problem in temporal graphs. Given a temporal graph $G^\tau = (V, E^\tau)$, an integer $k \leq n$, and a coloring function $c : V \to [k]$, the RainbowPath problem asks us to decide whether there exists a temporal path P^τ of length $k-1$ such that all vertex colors of P^τ are different. An example of RainbowPath problem is illustrated in Figure 2.

The RainbowPath problem is a special case of the PathMotif problem, where all the colors in the multi-set M are different, that is $M = [k]$. It is easy to see that the RainbowPath problem in static graphs can be reduced to the RainbowPath problem in temporal graphs by replacing each static edge with $k-1$ temporal edges.[3] So, the RainbowPath problem is **NP**-complete.

6 Algebraic algorithm for temporal paths

We now present an algorithm for the k-TempPath and PathMotif problems. Our algorithm relies on a *polynomial encoding* of temporal walks and the *algebraic fingerprinting* technique [6, 23, 26, 37]. The algorithm is presented in three steps: (i) we present a dynamic programming recursion to generate polynomial encoding of temporal walks; (ii) we present an algebraic algorithm to detect the existence of an multilinear monomial in the polynomial generated using the recursion in (i) — furthermore, we prove that existence of a multilinear monomial implies existence of a temporal path; (iii) finally, we extend the approach to detect temporal paths with additional color constraints using constrained multilinear detection. Let us begin with the concept of polynomial encoding of temporal walks.

Let P be a multivariate polynomial such that every monomial M is of the form $x_1^{d_1} x_2^{d_2} \ldots x_q^{d_q} y_1^{f_1} y_2^{f_2} \ldots y_r^{f_r}$. A monomial is *multilinear* if $d_i \in \{0, 1\}$ for all $i \in [q]$ and $f_j \in \{0, 1\}$ for all $j \in r$. A monomial is *x-multilinear* if $d_i \in \{0, 1\}$ for all $i \in q$, i.e., we do not take into account the degrees of the y-variables. The degree of a monomial M is the sum of the degrees of all its variables. However, for a x-multilinear monomial the degree is the sum of degrees of x-variables.

Monomial encoding of a temporal walk. Let

[3] Given a static graph $G = (V, E)$ and a coloring function $c : V \to [k]$, the RainbowPath problem in static graphs asks us to find a path P of length $k-1$ such that all vertex colors of P are different. The RainbowPath problem in static graphs is known to be **NP**-complete [13].

$W^\tau = v_1 e_1 v_2 \dots e_{k-1} v_k$ be a temporal walk in a temporal graph $G^\tau = (V, E^\tau)$. Let $\{x_{v_1}, \dots, x_{v_n}\}$ be a set of variables representing vertices in V and $\{y_{uv,\ell,i} : (u,v,i) \in E^\tau, \ell \in [k]\}$ be a set of variables such that $y_{uv,\ell,i}$ correspond to an edge $(u,v,i) \in E^\tau$ that appears at position ℓ in W^τ. A monomial encoding of W^τ is represented as

$$x_{v_1} y_{v_1 v_2,1,i_1} x_{v_2} y_{v_1 v_2,2,i_2} \cdots y_{v_{k-1} v_k,k-1,i_{k-1}}, x_{v_k},$$

where $i_1, \dots, i_{k-1}$ denote the timestamps on the edges $e_1, \dots, e_{k-1}$, respectively.

It can be shown that the above encoding of W^τ is x-multilinear if and only if W^τ is a temporal path.

Lemma 6.1. *The monomial encoding of a temporal walk W^τ is x-multilinear (and multilinear) if and only if the temporal walk is a temporal path.*

Generating polynomial for temporal walks. We present a recursion to generate temporal walks. Let $P_{u,\ell,i}$ denote the encoding of all walks of length $\ell - 1$ ending at vertex u and at latest time i. Let v_1 be a vertex such that $N_i(v_1) = \{v_2, v_3, v_4\}$. Let $P_{v_2,\ell-1,i-1}$, $P_{v_3,\ell-1,i-1}$ and $P_{v_4,\ell-1,i-1}$ represent the polynomial encoding of walks ending at vertices v_2, v_3 and v_4, respectively, such that all walks have length $\ell - 2$ and end at latest time $i-1$. Let $P_{v_1,\ell,i-1}$ denote polynomial encoding of all walks of length $\ell - 1$, ending at v_1 at latest time $i-1$. The example is illustrated in Figure 3.

The polynomial encoding to represent walks of length $\ell - 1$ ending at v_1 and at latest time i can be written as:

$$\begin{aligned} P_{v_1,\ell,i} \quad = \quad & x_{v_1} y_{v_2 v_1,\ell,i}\, P_{v_2,\ell-1,i-1} + \\ & x_{v_1} y_{v_3 v_1,\ell,i}\, P_{v_3,\ell-1,i-1} + \\ & x_{v_1} y_{v_4 v_1,\ell,i}\, P_{v_4,\ell-1,i-1} + P_{v_1,\ell,i-1}. \end{aligned}$$

Intuitively, the above equation represents that we can reach vertex v_1 at time step i if we have already reached any of its neighbors in $N_i(v_1)$ by latest timestamp $i-1$. Furthermore, the term $P_{v_1,\ell,i-1}$ is included so that if we have reached v_1 at latest time $i-1$ we can choose to stay at v_1 for timestamp i.

By generalizing the above idea, a generating function for $P_{u,\ell,i}$, for each $u \in V$, $\ell \in [k]$, and $i \in [t]$ is written as follows:

$$(6.1) \qquad P_{u,\ell,i} = x_u \sum_{v \in N_i(u)} y_{uv,\ell,i} P_{v,\ell-1,i-1} + P_{u,\ell,i-1}.$$

Furthermore, let us form the polynomial $\mathcal{P}_{\ell,i} = \sum_{u \in V} P_{u,\ell,i}$, for each $\ell \in [k]$ and $i \in [t]$. More precisely, $\mathcal{P}_{\ell,i}$ denotes the polynomial encoding of all walks of

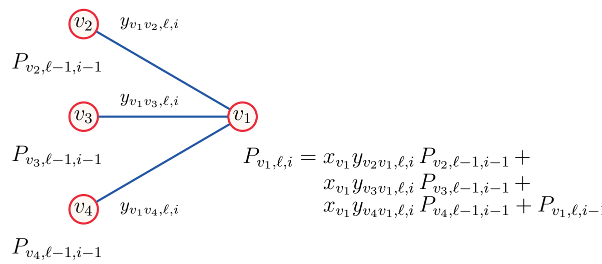


Figure 3: An illustration of the polynomial encoding of temporal walks.

length $\ell - 1$ ending at latest timestamp i. Now the problem of detecting a k-TempPath is equivalent to finding a x-multilinear monomial in $\mathcal{P}_{k,t}$. From the construction of the generating function in (6.1) it is clear that y variables are always distinct and detecting a x-multilinear monomial is equivalent to detecting a multilinear monomial.

Lemma 6.2. *The polynomial encoding $P_{u,\ell,i}$ in (6.1) contains a x-multilinear monomial of degree ℓ if and only if there exists a temporal path of length $\ell - 1$ ending at vertex u at latest time i.*

Multilinear sieving. From Lemma 6.2 the problem of deciding the existence of a k-TempPath in G^τ reduces to detecting the existence of a multilinear monomial term in $\mathcal{P}_{k,t}$.

Let L be the set of k labels and $[n]$ the set denoting vertices in V. For each vertex $i \in [n]$ and label $j \in L$ we introduce a new variable $z_{i,j}$. The vector of all variables of $z_{i,j}$ is denoted as $\mathbf{z}$ and the vector of all y-variables as $\mathbf{y}$. Now we can determine the existence of a multilinear monomial in $\mathcal{P}_{k,t}$ by making 2^k random substitutions of the new variables in $\mathbf{z}$ using the technique described by Björklund, Kaski and Kolwalik [5]. The algorithm is randomized and has a false negative probability of $\frac{2k-1}{2^b}$ where the arithmetic is over $\mathrm{GF}(2^b)$ [6].

Lemma 6.3. ([5]) *The polynomial $\mathcal{P}_{k,t}$ has at least one multilinear monomial if and only if the polynomial*

$$(6.2) \qquad Q(z, \mathbf{y}) = \sum_{A \subseteq L} \mathcal{P}_{k,t}(z_1^A, \dots, z_n^A, \mathbf{y})$$

is not identically zero, where $z_i^A = \sum_{j \in A} z_{i,j}$ for all $i \in [n]$ and $A \subseteq L$.

Algorithm. Our algorithm for k-TempPath works as follows: (i) we construct a polynomial representing all temporal walks of length $k-1$ using the recursion (6.1);

and (ii) we check if there exits a x-multilinear monomial term in the polynomial generated from (i) using Lemma 6.3. From Lemma 6.2, existence of a x-multilinear monomial term of size k implies existence of a temporal path of length $k-1$ and vice versa.

Lemma 6.4. *There exists an algorithm for solving the k-TempPath problem in $\mathcal{O}(2^k k(nt+m))$ time and $\mathcal{O}(nt)$ space.*

Lemma 6.5. *k-TempPath is fixed-parameter tractable.*

Constrained multilinear sieving. The previous section describes an algorithm for detecting k-TempPath. Now we discuss how to extend this approach to detect PathMotif using constrained multilinear sieving technique.

If we observe carefully, to obtain a PathMotif we need to find a multilinear monomial term in the polynomial $\mathcal{P}_{k,t}$ such that the vertex colors corresponding to the x-variables with degree one agrees to that of the multi-set M. This can be done by imposing additional constraints while evaluating the sieve.

Let C be a set of n colors and $c : [n] \to C$ a function that associates each $i \in [n]$ to a color in C. For each color $s \in C$ let us denote the number of occurrences of color s by $\mu(s)$. A monomial $x_1^{d_1} \dots x_q^{d_q} y_1^{f_1} \dots y_r^{f_r}$ is properly colored if for all $s \in C$ it holds that $\mu(s) = \sum_{i \in c^{-1}(s)} d_i$, more precisely the number of occurrences of color s is equal to the total degree of x-variables representing the vertices with color s.

For each $s \in C$, let S_s be the set of $\mu(s)$ with color s such that $S_s \cap S_{s'} = 0$ for all $s \neq s'$. For $i \in [n]$ and $d \in S_{c(i)}$ we introduce a new variable $v_{i,d}$. Let L be a set of K labels. For each $d \in \cup_{s \in C} S_s$ and each label $i \in L$ we introduce a new variable $w_{i,d}$.

Lemma 6.6. ([6]) *The polynomial $\mathcal{P}_{k,t}$ has at least one monomial that is both x-multilinear and properly colored if and only if the polynomial*

$$Q(z, \mathbf{w}, \mathbf{y}) = \sum_{A \subseteq L} \mathcal{P}_{k,t}(z_1^A, \dots, z_n^A, \mathbf{y}) \tag{6.3}$$

is not identically zero, where

$$z_i^A = \sum_{j \in A} z_{i,j}, \quad \text{and} \quad z_{i,j} = \sum_{d \in S_{c(i)}} v_{i,d} w_{d,j}. \tag{6.4}$$

From Lemmas 6.4 and 6.6 it follows that we have an $\mathcal{O}(2^k k(nt+m))$ algorithm to solve PathMotif problem in $\mathcal{O}(nt)$ space.

Obtaining an optimal solution. In this section we describe a procedure to obtain an optimal path. By optimal we mean that the maximum timestamp of the edges in the temporal path is minimized. For simplicity, we refer to our algorithm for the decision version as *decision oracle*. To find the minimum (optimal) timestamp $t' \in [t]$, we make at most $\mathcal{O}(\log t)$ queries to the decision oracle using binary search on range $[t]$.

Extracting a solution. In the previous sections we described an algebraic solution for the decision version of the PathMotif problem. In many cases we need to extract a solution, if such a path exists. We use the decision oracle as a subroutine to find a solution in at most $\mathcal{O}(n)$ queries as follows: (i) for each vertex $v \in V$ we remove the vertex v and the edges incident to it and query the oracle. If there is a solution, then we continue to next vertex; otherwise we put back v and the edges incident to it, and continue to next vertex. In this way, we can obtain a subgraph with k vertices in at most $n-k$ queries to the oracle. However, the number of queries to the decision oracle can be reduced to $\mathcal{O}(k \log n)$ queries in expectation by recursively dividing the graph in to two halves [5]; (ii) pick an arbitrary start vertex in the subgraph obtained from (i) and find a temporal path connecting all the k vertices using temporal DFS, if such a path do not exist then continue to next vertex. Even though the worst case complexity is $\mathcal{O}(k!)$, in practice this approach works very fast. However, extracting a solution can be done using $\mathcal{O}(k)$ queries to the decision oracle using vertex-localized sieving. For the reasons of space we skip a detailed discussion of this approach.

Path motif problem with delays. In a real-world transport network a transition between any two locations would involve a *transition time* and a minimum *delay time* at a location before continuing the journey, for example a minimum time to visit a museum. In this section, we introduce a problem setting with transition and delay times and present generating polynomials to solve the problems.

For a temporal graph $G^\tau = (V, E^\tau)$, an edge $e \in E^\tau$ is a quadruple (u, v, i, Δ) where $u, v \in V$, $i \in \mathbb{Z}_{\geq 0}$ is a time instance and $\Delta \in \mathbb{Z}_{\geq 0}$ is transition time from u to v. Additionally, each vertex has a delay time $\delta : V \to \mathbb{Z}_{\geq 0}$. The encoding with transition time and delay is the following:

$$P_{u,\ell,i+\Delta} = x_u \sum_{(u,v,i,\Delta) \in E} y_{uv,i} P_{v,\ell-1,i-\delta(v)} + P_{u,\ell,i-1}.$$

From Lemmas 6.2 and 6.6, it follows that existence of a multilinear monomial in the polynomial generated above would imply the existence of a PathMotif.

7 Implementation

We use the design of Björklund et al. [7] as a starting point for our implementation, in particular we make use

Table 1: Comparison of extraction time for baseline and algebraic algorithms. All runtimes are in seconds.

m	Regular			Powlaw $d^{-0.5}$			Powlaw $d^{-1.0}$		
	Base	Alg	Sp	Base	Alg	Sp	Base	Alg	Sp
1040	0.1	0.1	1	0.1	0.1	1	0.1	0.1	1.0
10040	0.5	0.1	5	1.0	0.1	10	10.8	0.1	**108**
100040	5.6	1.1	5	30.4	1.1	27.6	20430.2	1.0	**20430.2**
1000040	74.0	12.0	6	808.2	11.2	**72.1**	–	10.1	–

of fast finite-field arithmetic implementation.

Our effort boils down to implementing the generating function (6.1) and evaluating the recurrence at 2^k random points. Specifically, we introduce a domain variable x_v for each $v \in V$ and a support variable $y_{uv,\ell,i}$ for each $\ell \in [k]$ and $(u, v, i) \in E^\tau$. The values of variables x_v are computed using Equation (6.4) and the values of variables $y_{uv,\ell,i}$ are assigned uniformly at random.

Our current implementation uses $\mathcal{O}(ntk)$ memory instead of $\mathcal{O}(nt)$. In order to reduce the memory access latency we arrange our memory layout as $k \times t \times n$; furthermore, we employ hardware prefetching [7, § 3.6] to saturate the memory bandwidth and a parallelization scheme [7, § 3.5] to achieve thread-level parallelism.

Our software is available as open source [33].

8 Experimental evaluation

In this section we discuss our experimental evaluation.

Baseline. For the problems considered in this paper we are not aware of any known baselines to compare. Thus, we implemented two baselines: (i) an *exhaustive-search* algorithm using temporal DFS, and (ii) a *brute-force* algorithm based on random walks. The details of these algorithms are available in an extended abstract [34]. The brute-force algorithm does not work in practice, even for small graphs ($m = 10^4$). For this reason, we experiment only with the exhaustive-search baseline. We note that the baseline is highly optimized and thread parallelized.

Hardware. We experiment with two configurations.
Workstation: A Fujitsu Esprimo E920 with 1×3.2 GHz Intel Core i5-4570 CPU, 4 cores, 16 GB memory, Ubuntu, and `gcc` v 5.4.0.
Computenode: A Dell PowerEdge C4130 with 2×2.5 GHz Intel Xeon 2680 V3 CPU, 24 cores, 12 cores/CPU, 128 GB memory, Red Hat, and `gcc` v 6.3.0.

Our executions make use of all cores.

Input graphs. We evaluate our methods using both synthetic and real-world graphs. (*i*) We use two types of *synthetic graphs*: random d-regular graphs; and power-law graphs. (*ii*) We use the *real-world* road transport networks from the cities of Helsinki and Madrid. A description of datasets and graph configuration models is available in an extended version of this paper [34].

Our baseline and scalability experiments are performed on RainbowPath problem instances, remember that, in RainbowPath problem every vertex matches with a multi-set color. Likewise, no trivial preprocessing step can be employed to reduce the graph size.

8.1 Experimental results.

We now describe our results and key findings. Recall that *decision time* is the time required to decide the existence of one solution, while *extraction time* is the time required to extract such a solution. As discussed previously, extracting a solution requires multiple calls to the decision oracle. All the experiments are executed on the workstation using all cores, with an only exception for the experiments with scalability to large graphs which is executed on the computenote.

Baseline. Our first set of experiments compares the *extraction* time to obtain an optimal solution using our algebraic algorithm and the exhaustive-search baseline. In Table 1, we report extraction times for: (*i*) *d-regular* random graphs with $n = 10^2, \ldots, 10^5$ and fixed values of $d = 20$, $t = 100$, $k = 5$; (*ii*) *power-law* graphs with $n = 10^2 \ldots, 10^5$, $D = 20$, $w = 100$, $k = 5$, $\alpha = -0.5$; and (*iii*) $\alpha = -1.0$. Vertex colors are assigned randomly in the range $[k]$ and the multi-set is $[k]$. Each graph instance has at least ten target instances agreeing multi-set colors with different timestamps chosen uniformly at random. For the baseline we report the *minimum* time of five independent runs, however, for the algebraic algorithm we report the *maximum*. *Speedup* (Sp) is the ratio of baseline and algebraic algorithm runtimes.

Surprisingly, the baseline can compete with the algebraic algorithm in the case of d-regular random graphs, however, the runtimes have high variance. On the other hand, the algebraic algorithm is very stable. For the power-law graphs with $m = 10^5$ edges and multi-set size $k = 5$, the algebraic algorithm is at least *twenty thousand* times faster than the baseline. The baseline failed to report a solution in small graphs $m = 10^3$ with large multi-set size $k = 10$.

Scalability. Our second set of experiments study scalability with respect to: (*i*) number of edges; (*ii*) multi-set size; (*iii*) number of timestamps; and (*iv*) vertex degree.

Figure 4 (left) reports decision and extraction times for d-regular random graphs with $n = 10^2, \ldots, 10^5$ and fixed values of $d = 20$, $k = 8$, $t = 100$. Figure 4 (center-left) shows decision time for d-regular random graphs with $k = 10, \ldots, 18$ and fixed values of $n = 10^3$, $d = 20$, $t = 100$. Vertex colors are assigned randomly in the range $[k]$ and the multi-set is $[k]$. We observe a

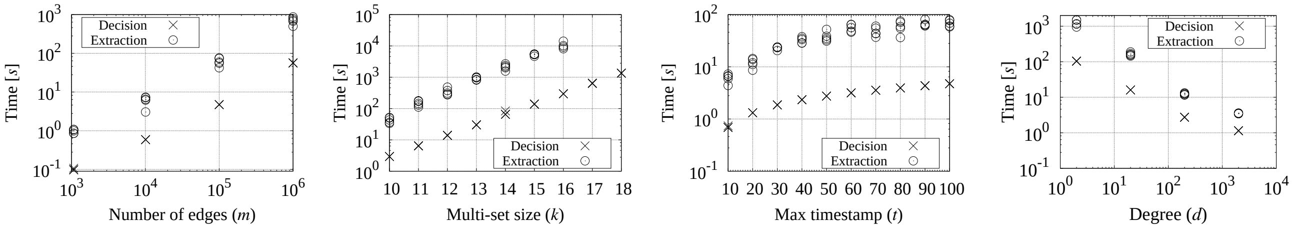


Figure 4: Scalability results. Runtime as a function of the number of edges (left); multi-set size (center-left); number of timestamps (center-right); and degree (right).

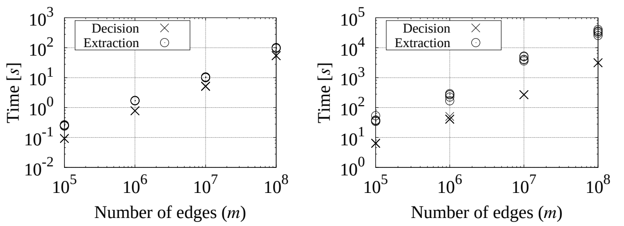


Figure 5: Scaling to large graphs. Runtime as a function of number of edges with $k = 5$ (left) and $k = 10$ (right).

Table 2: Experimental results on real-world graphs. All runtimes are in seconds. H–Helsinki, M–Madrid.

Dataset	n	m	t	$k=5$		$k=10$	
				Base	Alg	Base	Alg
Tram(M)	70	35144	1265	1.37	0.24	1337.98	28.05
Train(M)	91	43677	1181	40.01	0.25	–	24.12
Bus(M)	4597	2254993	1440	6337.89	1.27	–	278.91
IU-bus(M)	7543	1495055	1440	744.79	1.30	–	325.51
Bus(H)	7959	6403785	1440	–	1.67	–	444.66

linear scaling with increasing the number of edges and exponential scaling with increasing the multi-set size, as expected by the theory. The variance in decision time is very small for different inputs, however, it is higher for extraction time. The algorithm is able to decide the existence of a solution in less than two minutes for graphs up to one million edges with multi-set size $k = 8$ and extract a solution in less than sixteen minutes.

Next we study the effect of graph density on scalability. Figure 4 (center-right) shows decision and extraction times for d-regular random graphs with $t = 10, \ldots, 100$ and fixed values of $n = 10^4$, $d = 20$, $k = 8$. Figure 4 (right) shows decision and extraction times for d-regular random graphs with $d = 2, 20, 200, 2000$ and corresponding values of $n = 10^6, \ldots, 10^3$, with fixed $m = 10^6$ and $t = 100$. We observe that the algebraic algorithm performs better for dense graphs. A possible explanation is that for sparse graphs there is not enough work to keep both the arithmetic and memory pipeline busy, simultaneously.

Scaling to large graphs. Next we study the scalability of the algebraic algorithm to graphs with up to hundred million edges. Figure 5 reports decision and extraction times for d-regular random graphs with $n = 10^3, \ldots, 10^6$, $d = 200$, $t = 100$ with $k = 5$ (left) and $k = 10$ (right). Vertex colors are assigned randomly in the range $[k]$ and the multi-set is $[k]$. In graphs with hundred million edges, the algebraic algorithm can extract an optimal solution in less than two minutes for $k = 5$ and less than two hours for $k = 10$.

Experiments with real-world graphs. Finally, we report decision and extraction times for the algebraic algorithm on real-world data. Table 2 reports decision and extraction time (in seconds) for the experiments on real-world datasets. For each dataset we report the maximum time among the five independent executions by choosing multi-set colors at random. For multi-set size $k = 5$, the extraction time is at most two seconds. For larger multi-set size $k = 10$, the extraction time is at most eight minutes in all the datasets. Additionally, we pre-process the graphs by removing vertices whose colors do not match with multi-set colors for both the baseline and the algebraic algorithm.

9 Conclusions and future work

In this paper we introduce several pattern-detection problems that arise in the context mining large temporal graphs. We present complexity results, and design algebraic algorithms based on the constrained multilinear sieving technique. Our implementation can scale to large graphs up to hundred million edges despite the problems being **NP**-hard. We present extensive experimental results that validate our scalability claims.

As a future work we would like to consider problem settings where we search for temporal arborescences and temporal subgraphs. Furthermore, we would like to explore the counting variants of these problems.

References

[1] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp, *Biomolecular network motif counting and discovery by color coding*, Bioinformatics, 24 (2008), pp. 241–249.

[2] C. Aslay, A. Nasir, G. De Francisci Morales, and A. Gionis, *Mining frequent patterns in evolving graphs*, in CIKM, 2018, pp. 923–932.

[3] A. Benson, D. Gleich, and J. Leskovec, *Higher-order organization of complex networks*, Science, 353 (2016), pp. 163–166.

[4] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, *Narrow sieves for parameterized paths and packings*, JCSS, 87 (2017), pp. 119–139.

[5] A. Björklund, P. Kaski, and Ł. Kowalik, *Determinant sums for undirected Hamiltonicity*, SIAM J. Comput., 43 (2014), pp. 280–299.

[6] A. Björklund, P. Kaski, and L. Kowalik, *Constrained multilinear detection and generalized graph motifs*, Algorithmica, 74 (2016), pp. 947–967.

[7] A. Björklund, P. Kaski, L. Kowalik, and J. Lauri, *Engineering motif search for large graphs*, in ALENEX, 2015, pp. 104–118.

[8] M. Bressan, S. Leucci, and A. Panconesi, *Motivo: Fast motif counting via succinct color coding and adaptive sampling*, PVLDB, 12 (2019), pp. 1651–1663.

[9] A. Casteigts, A. Himmel, H. Molter, and P. Zschoche, *The computational complexity of finding temporal paths under waiting time constraints*, CoRR, abs/1909.06437 (2019).

[10] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized algorithms*, 2015.

[11] M. DeChoudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu, *Automatic construction of travel itineraries using social breadcrumbs*, in HT, 2010, pp. 35–44.

[12] H. Dell, J. Lapinskas, and K. Meeks, *Approximately counting and sampling small witnesses using a colourful decision oracle*, in SODA, 2020, pp. 2201–2211.

[13] E. Eiben, R. Ganian, and J. Lauri, *On the complexity of rainbow coloring problems*, DAM, 246 (2018), pp. 38 – 48.

[14] T. Gagie, D. Hermelin, G. M. Landau, and O. Weimann, *Binary jumbled pattern matching on trees and tree-like structures*, in ESA, 2013.

[15] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 29, W. H. Freeman and Co., 2002.

[16] B. George, S. Kim, and S. Shekhar, *Spatio-temporal network databases and routing algorithms: A summary of results*, in SSTD, 2007, pp. 460–477.

[17] E. Giaquinta and S. Grabowski, *New algorithms for binary jumbled pattern matching*, IPL, 113 (2013), pp. 538–542.

[18] A. Gionis, T. Lappas, K. Pelechrinis, and E. Terzi, *Customized tour recommendations in urban areas*, WSDM, 2014, pp. 313–322.

[19] M. Gupta, C. C. Aggarwal, and J. Han, *Finding top-k shortest path distance changes in an evolutionary network*, in SSTD, 2011, pp. 130–148.

[20] P. Holme and J. Saramäki, *Temporal networks*, Physics reports, 519 (2012), pp. 97–125.

[21] ———, *Temporal networks*, Physics reports, 519 (2012), pp. 97–125.

[22] P. Kaski, J. Lauri, and S. Thejaswi, *Engineering Motif Search for Large Motifs*, in SEA, 2018, pp. 1–19.

[23] I. Koutis, *Faster algebraic algorithms for path and packing problems*, in ICALP, 2008.

[24] ———, *The power of group algebras for constrained multilinear monomial detection*, Dagstuhl meeting 10441, (2010).

[25] ———, *Constrained multilinear detection for faster functional motif discovery*, IPL, 112 (2012), pp. 889–892.

[26] I. Koutis and R. Williams, *Limits and applications of group algebras for parameterized problems*, in ICALP (1), 2009.

[27] I. Koutis and R. Williams, *Algebraic fingerprints for faster algorithms*, Comm. of the ACM, 59 (2016), pp. 98–105.

[28] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, *Temporal motifs in time-dependent networks*, JSM, 2011 (2011), p. P11005.

[29] L. Kowalik and J. Lauri, *On finding rainbow and colorful paths*, TCS, 628 (2016), pp. 110 – 114.

[30] M. Latapy, T. Viard, and C. Magnien, *Stream graphs and link streams for the modeling of interactions over time*, Social Network Analysis and Mining, 8 (2018).

[31] P. Liu, A. Benson, and M. Charikar, *Sampling methods for counting temporal motifs*, in WSDM, 2019, pp. 294–302.

[32] A. Paranjape, A. Benson, and J. Leskovec, *Motifs in temporal networks*, WSDM, 2017, pp. 601–610.

[33] S. Thejaswi and A. Gionis, 2019. `https://github.com/suhastheju/temporal-patterns`.

[34] ———, *Finding temporal patterns using algebraic fingerprints*, arXiv, abs/2001.07158 (2020).

[35] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, *The orienteering problem: A survey*, EJOR, 209 (2011), pp. 1 – 10.

[36] B. Wackersreuther, P. Wackersreuther, A. Oswald, C. Böhm, and K. Borgwardt, *Frequent subgraph discovery in dynamic networks*, in MLG, 2010.

[37] R. Williams, *Finding paths of length k in $O^*(2^k)$ time*, IPL, 109 (2009).

[38] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, *Path problems in temporal graphs*, Proc. VLDB Endow., 7 (2014), pp. 721–732.

[39] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu, *Efficient algorithms for temporal path computation*, TKDE, 28 (2016), pp. 2927–2942.

[40] J. Yang, J. McAuley, and J. Leskovec, *Community detection in networks with node attributes*, in ICDM, 2013, pp. 1151–1156.