Theoretical and Empirical Analysis of the Representability of Dynamic Networks as Edge Periodic Temporal Graphs

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Universität Trier
FB IV - Informatikwissenschaften
Lehrstuhl für Theoretische Informatik

Gutachter*in:	Prof. Dr. Henning Fernau
	Dr. Petra Wolf
Betreuerin:	Dr. Petra Wolf

Vorgelegt am 10. Januar 2024 von:

Sven Fiergolla
Kopernikusstr 23
10245 Berlin
sven.fiergolla@gmail.com
Matr.-Nr. 1252732

Abstract

Edge periodic temporal graphs (EPGs) are graphs where edge availability varies over time, that can represent the periodic behavior of dynamic networks. In EPGs, for each edge e of the graph, a binary string s_e determines in which time steps the edge is present, namely, e is present in time step t if and only if s_e contains a 1 at position $t \ mod \ |s_e|$. Due to their periodic nature, EPGs can be a more concise way of representing complex periodic systems compared to traditional temporal graphs. In this thesis, we study the effectiveness of representing dynamic networks as EPGs theoretically and empirically, by interpreting the labels as unary automata and applying a decomposition algorithm for DFAs as well as two novel algorithms that identify other, more "explainable" factors for human interpreters.

Übersetzung

Edge-periodische Zeitgraphen (EPGs) sind Graphen, deren Kantenverfügbarkeit sich im Laufe der Zeit ändert und die das periodische Verhalten dynamischer Netzwerke darstellen können. In EPGs bestimmt für jede Kante e des Graphen eine binäre Zeichenkette s_e, in welchen Zeitschritten die Kante vorhanden ist; das heißt, e ist im Zeitschritt t vorhanden, wenn und nur wenn s_e an Position $t \ mod \ |s_e|$ eine 1 enthält. Aufgrund ihrer periodischen Natur können EPGs eine prägnantere Möglichkeit darstellen, komplexe periodische Systeme im Vergleich zu herkömmlichen zeitlichen Graphen zu repräsentieren. In dieser Arbeit untersuchen wir theoretisch und empirisch die Effektivität der Darstellung dynamischer Netzwerke als EPGs, indem wir die Labels als unäre Automaten interpretieren und einen Dekompositionsalgorithmus für DFAs anwenden, sowie zwei neuartige Algorithmen, die andere, für menschliche Interpreten potenziell "erklärbarere" Faktoren identifizieren.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Temporal graphs are graphs where each edge is associated with a binary string that indicates in which time-step the edge is present. They allow us to identify and capture recurring patterns in temporal data. By detecting periodicity, we can gain insights into regular behaviors or events that occur at fixed intervals. This can be valuable in fields such as finance, transportation, and social networks, where understanding periodic trends can aid in decision-making and prediction. Temporal graphs can become quite complex as they evolve over time and labels for each edge can have different lengths, potentially resulting in the entire sequence repeating only after an exponential period. By transforming them into edge periodic temporal graphs, we can simplify the analysis of e.g. finding regularities by focusing on the repeating patterns instead of dealing with the entire temporal range. This reduction in complexity can lead to more efficient algorithms and computational benefits. In scenarios where resources need to be allocated or scheduled based on temporal patterns, edge periodic temporal graphs enable more efficient usage. For example, optimizing transportation routes based on recurring traffic patterns can significantly reduce congestion and improve resource allocation. Furthermore from a data compression standpoint, the conversion leads to more compact representations. Rather than storing the entire temporal history, edge periodic temporal graphs capture the essential periodic patterns, reducing storage and memory requirements while preserving critical information. In conclusion, the transformation of temporal graphs into edge periodic temporal graphs serves as a powerful technique that unlocks a variety of applications.

Imagine standing at a train station connecting place A to place B, departing every half and full hour. The schedule gives information about a leaving train for any time-step t for intervals of 5 minutes. To cover all possible time-steps in a day, a long index with all steps is listed, where you have to search for your exact arrival time, to see when the next train is leaving, as this is not apparent in the large table with many additional time points. Now imagine the schedule in a different representation, which is periodically repeating after one hour. To find the next train, one just has to check at which time of an hour a train departs, which is not only more concise but also easier to understand for human interpreters.

1.2 Problem Statement

To use the benefits of the periodic representation as an edge periodic graph, a transformation from temporal to edge periodic has to be performed. This transformation

can be done in different ways, but we focus on a method based on the decomposition of a deterministic finite automaton. Reading the labels of a temporal graph and interpreting it as DFA enables the application of known methods for DFA decomposition. Labels from a temporal graph, consisting of binary strings, can be interpreted as unary DFAs by representing the 1 values as final states of a DFA.

Temporal graph with long edge labels EPG with short and periodic labels

Figure 1.1: Different temporal graph types

Then the DFA decomposition algorithms can be applied, to get a family of smaller DFAs, which combined resembles the language of the original DFA. These so-called factors each represent a periodic part of the input and can be used to get a shorter and periodic label as visible in Figure 1.1. Since we obtain a decomposition family, we might need to replace the original temporal graph with an edge periodic graph with multi-edges.

1.3 Structure of this Work

This work is structured into a first introduction to notation and a survey of the current state of the art of temporal and edge periodic graph analysis as well as the decomposition of DFAs. Following this, the used real-world data sets are described, in particular the F2F graphs. The core of the document revolves around the decomposition of edge labels by decomposing DFAs. Section 3 explains the decomposition of DFAs and unary DFAs, while Section 3.2 describes a LogSpace decomposition algorithm, employing maximal divisors. A dedicated Section 4 compares AND vs OR decomposition. The aspect of "explainability" and explainable edge labels or decompositions for human analysis is thoroughly examined in Section 5. This includes the introduction of an explainability metric and the methodology for measuring explainability. Novel approaches to the problem are presented in Section 6, including Greedy Short Factors and Fourier Transform. Additionally, a problem generalization is formally analyzed. The subsequent sections detail the implementation of these approaches in Section 7 and their evaluation in Section 8. The evaluation includes both performance and decomposition assessments, with a breakdown of different methods of decomposition. Furthermore, the document assesses the explainability of the proposed methods. Finally, the document concludes with a discussion in Section 9 where the findings are contextualized, and implications are explored.

2. Preliminaries

2.1 Notation

We denote by $\mathbb{N}$ the set of non-negative integers $\{0, 1, 2, \dots\}$. We call $d \in \mathbb{N}$ a maximal divisor of an integer $i \in \mathbb{N}$, if $d|i$ and there exists no $c \in \mathbb{N}$ with $c|d$ and $c|i$.

For a word $w = w_1 w_2 \dots w_n$ with $w_i \in \sum$ for $1 \leq i \leq n$, $w[i]$ represents the symbol w_i at position i in w and $\ell(w)$ be the length of w. We write the concatenation of strings u and v as $u \cdot v$. Moreover, for every $\sigma \in \Sigma$, we denote by $\#\sigma(w)$ the number of times the letter σ appears in w and let w^n be the concatenation of w with itself, n times. The bit wise or operation on binary words is denoted as $\vee$, the bit-wise and operation as $\wedge$ such that $a = b \vee c$ with, for $1 < i < \max \ell(b), \ell(c)$, $a[i] = b[i] \vee c[i]$ if $\min \ell(b), \ell(c) > i$ and $a[i] = b[i]$ (respectively $c[i]$), if $\ell(c) < i$ (respectively $\ell(b) < i$).

A deterministic finite automaton (DFA) is a 5-tuple $A = \{\Sigma, Q, q_I, \sigma, F\}$, where Σ is a finite non-empty alphabet, Q is a finite set of states, $\sigma : Q \times \Sigma \to Q$ is a transition function, $q_I \in Q$ is the initial state, and $F \subseteq Q$ is a set of accepting states. The states in $Q \backslash F$ are called rejecting states. The transition function σ is expanded to words by defining it recursively $\sigma : Q \times \Sigma^* \to Q$ is $\sigma(q, \epsilon) = q$ and $\sigma(q, w_1, w_2 \dots w_n) = \sigma(\sigma(q, w_1 w_2 \dots w_{n-1}), w_n)$. The *run* of A on a word $w = w_1 \dots w_n$ is the sequence of states $s_0, s_1, \dots, s_n$ such that $s_0 = q_I$ and for each $1 \leq i \leq n$ it holds that $\sigma(s_{i-1}, w_i) = s_i$. Note that $s_n = \sigma(q_I, w)$. The DFA A *accepts* w iff $\sigma(q_I, w) \in F$. Otherwise, A *rejects* w. The set of words accepted by A is denoted $L(A)$ and is called the *language* of A. A language accepted by some DFA is called a *regular language*. We refer to the size of a DFA A, denoted $|A|$, as the number of states in A. A DFA A is *minimal* if every DFA B such that $L(B) = L(A)$ satisfies $|B| \geq |A|$. We call a DFA a *permutation* DFA if for each letter $\sigma \in \Sigma$, the transition function is a bijection and we call a DFA a *commutative* DFA if $\sigma(q, uv) = \sigma(q, vu)$ for every state q and every pair of words $u, v \in \Sigma^*$.

We call a DFA A *and-composite* if there exists a family $(B_i)_{1 \leq i \leq k}$ of DFAs with $|B_i| < |A|$ for all $1 \leq i \leq k$, such that $L(A) = \cap_{1 \leq i \leq k} L(B_i)$ and call the family $(B_i)_{1 \leq i \leq k}$ a *AND-decomposition* of A. Note that, all B_i in the decomposition satisfy $|B_i| < |A|$ and $L(A) \subseteq L(B_i)$. Such DFAs are called *factors* of A, and $(B_i)_{1 \leq i \leq k}$ is also called a k-factor decomposition of A. The width of A is the smallest k for which there is a k-factor decomposition of A, and we say that A is k-factor composite iff $width(A) \leq k$. We call a DFA A prime if it is not composite. Similarly, we call A or-composite, if there exists a family $L(A) = \cup_{1 \leq i \leq k} L(B'_i)$ of DFAs with $|B'_i| < |A|$ and we call $\cup_{1 \leq i \leq k} L(B'_i)$ an OR-decomposition of A.

When analyzing an OR-decomposition $(B_i)_{1 \leq i \leq k}$, a partial OR-decomposition up to $j < k$ might have words which are accepted by A but not by $(B_i)_{1 \leq i \leq j}$, those are

called *outliers*. Let $A = (\Sigma, Q, q_I, \sigma, F)$. We define $out(\{B_1, B_2, \ldots, B_j\}) = \{q \in F \mid \forall w \in \Sigma^* \colon \sigma(q_I, w) = q \rightarrow w \notin \bigcup_{1 \leq i \leq j} L(B_i)\}$.

An *edge periodic temporal graph* (EPG), $G = (V, E, \tau)$ consists of a graph $G = (V, E)$ called the underlying graph and a function $\tau : E \rightarrow \{0, 1\}^*$ where τ maps each edge e to a string $\tau(e) \in \{0, 1\}^*$, such that e exists in a time step $t \geq 0$ if and only if $\tau(e)[t] = 1$, where $\tau(e)[t] := \tau(e)[t \ mod \ |\tau(e)|]$. We denote the subgraph of G in time step t with $G(t)$ and call it the *snapshot graph* at time step t.

2.2 State of the Art

Temporal graphs, also known as time-varying or dynamic graphs, recently gained significant attention. Various aspects like their representation [1], analysis [2][3] as well as mining [4] have been explored. Techniques for community detection, aiming to identify groups of nodes which show cohesive patterns of interactions over time, have been developed by Al-sharoa et al. in [5]. Link prediction techniques have also been explored to predict future edge occurrences based on historical data [7]. Additionally, various metrics have been proposed to quantify important properties of temporal graphs, such as temporal centrality and clustering coefficients [8]. Understanding the evolution and dynamics of temporal graphs is another research area. Models have been developed to simulate the evolution of temporal graphs [9], considering factors such as edge arrival, departure, and temporal dependencies. Temporal network analysis involves the development of measures and techniques to characterize and visualize temporal graphs. Visualization techniques have been designed to effectively represent and explore the dynamic nature of temporal graphs [10]. Additionaly, temporal graph research finds applications in diverse fields. For example, it has been applied to study social networks, analyzing the evolution of friendships, communities, and information diffusion [11][12]. In transportation systems, temporal graphs help model traffic patterns, optimize routes, and analyze network dynamics [13]. In the realm of biology, temporal graphs have been used to study protein-protein interaction networks [14], gene regulatory networks, and other biological systems to understand dynamic processes [15]. Overall, the study of temporal graphs is a multidisciplinary field, with researchers from computer science, biology, physics, and other domains contributing to its advancements. Ongoing research continues to explore new techniques and methodologies for analyzing, modeling, and understanding the complex dynamics of temporal graphs.

Contrary to the well-established and studied field of temporal graphs, edge periodic graphs have received less attention. EPG Research has explored areas such as mining periodic cliques [16], identifying minors and subgraphs in EPGs [17], and analyzing EPGs within the context of the Game of Cops and Robbers [18]. However, the transformation of temporal graphs into EPGs remains an unexplored area. On the other hand, there are various resources regarding the study of the decomposition of different types of deterministic finite automatons, especially the problem:

Definition 2.2.1 (DFA Decomposition)**.**
Input: A DFA $\mathcal{A}$
Question: Is $\mathcal{A}$ *and-composite?*

and also the bounded variant of the problem which asks whether a DFA is k-factor composite. While initially the decomposition was proposed for general DFAs

in [19], algorithms and complexity bounds for the unbounded decomposition for unary DFAs were found by Jecker, Kupferman and Mazzocchi [20] and for bounded decomposition by Jecker, Mazzocchi and Wolf in [21]. Further detail about their works in Section 3.2 and in Figure 2.1. The proofs are formalized for unary DFAs but they can also be applied to binary labels as transitions are trivial and either 1's or 0's can represent the final states F which will be discussed in Section 3.1.

	DECOMP	Bounded k-DECOMP
DFA	EXSPACE [19]	PSPACE [21]
Unary DFA	LOGSPACE [20]	LOGSPACE [21]

Table 2.1: Complexity of unbounded and bounded DFA Decomposition

2.3 Real World Data (F2F Graphs)

Real world data is helpful in evaluating the effectiveness of the decomposition, therefore a collection of temporal graphs has been gathered from the Stanford Large Network Dataset Collection[1]. All the temporal graph datasets except one, contain interaction networks of forums such as Stackoverflow or Reddit and are in a format, where the edges have timestamps, at which point in time they are active. The Reddit network includes timestamps in Unix epoch seconds over a period of 3 years, consequently, the edges are only very rarely present in the graph. However, it would be possible to reinterpret these, for example, by increasing the time span between discrete time points. Instead of a time span of 1 second between events, it would also be possible to set the time span to one day, and a label at time point t would contain a 1 if on that day, person A commented on person B's post. We opted to used the Dynamic Face-to-Face Interaction Networks (F2F) dataset, where the data format is very close to the formal definition, containing binary labels with discrete steps and a one at step t indicating the existence of the respective edge. These networks represent the interactions that happen during discussions between a group of participants playing a game called Resistance. This dataset contains networks extracted from 62 games, each game is played by 5-8 participants and lasts between 45-60 minutes. They extracted dynamically evolving networks from the free-form discussions

[1] https://snap.stanford.edu/data/

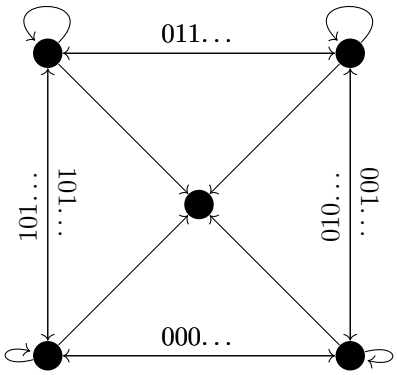
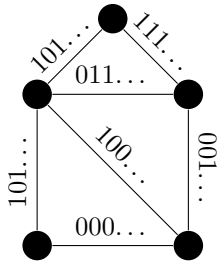


Figure 2.1: Example of the real word input data (F2F Files)

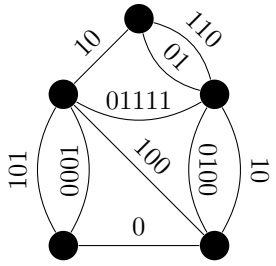
using the ICAF algorithm [22]. The extracted networks are used to characterize and detect group deceptive behavior using the DeceptionRank algorithm [23]. The networks are unweighted, directed and temporal, each node represents a participant. At each 1/3 second, a directed edge from node u to v indicates participant u looks at participant v or the laptop. There are also weighted versions of the networks for other purposes and more of a social analysis but we focus on the unweighted versions as they fit the definition of temporal graphs best.

3. Decomposition of Edge Labels

As our goal is to represent general dynamic networks and temporal graphs as EPGs, one problem is the missing periodicity in general temporal and dynamic graphs.



Temporal graph with long labels



EPG with multi edges but short labels

Figure 3.1: Desired transformation of temporal graphs into EPGs

To transform the long, confusing schedule from the example in Section 1.1 into a periodic and clear schedule, the temporal graph needs to be broken down and transformed into an edge periodic graph, see Figure 3.1. To transform a temporal graph, for each label the shortest possible string w' which is equal to the original label at every time step $\forall t \geq 0, \tau(e)[t] = w'[t]$ has to be found. This w' can itself be composed by combining multiple words $w_1, w_2, \ldots, w_n$ of different length, with $w'[t] = 1$ if there is an w_i with $w_i[t \ mod \ \ell(w_i)] = 1$ which represent the periodic components of the original label. Representing it as graph, the edges become multi edges, one for each word representing the original label. Initially, we start with a temporal graph where all edge labels $\tau(e)$ have fixed length and there are no periods present. To find and analyze such words in the given labels, the label is interpreted as a DFA, and an algorithm for decomposing DFAs is adapted to our problem to find smaller DFAs called factors. To apply the algorithm that is defined for automata, a label $w \in \{0,1\}^*$ is interpreted as an unary DFA. In the label either the $0s$ or the $1s$ symbols are used to represent final states F, see Figure 3.2. The algorithm from [21] which is also described in closer detail in Section 3.2 as well as two novel algorithms are used for decomposing DFAs. We only represent unary automata with $|\Sigma| = 1$ and therefore only have a single transition from each state, basically forming a simple cycle of all possible states. This means that for a period length of i we only have to check i states on the cycle being in the same state. Decomposing the obtained DFA results in a family of smaller DFAs, with $\cap_{i=1}^{k} L(\mathcal{A}_i)$ which can be interpreted as a set of binary labels w_i and their bitwise intersection $\wedge_{i=1}^{k} w_i^{\ell(w')/\ell(w_i)}$ equals w'. Either the bit-wise intersection over all found words is assigned the new label, or each word obtained by interpreting a factor can be its own label, forming an edge periodic multi-graph.

3.1 Decomposition of DFAs and Unary DFAs

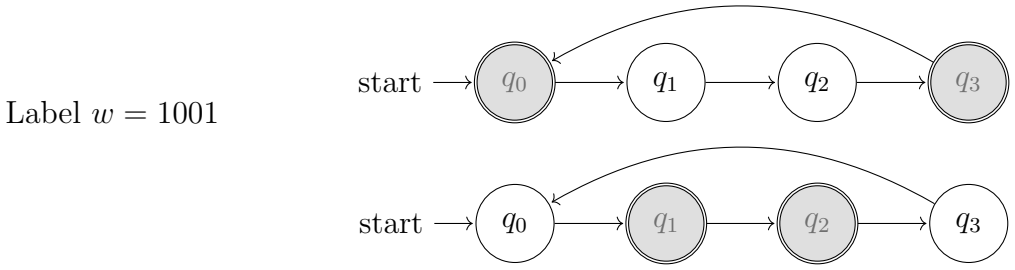


Figure 3.2: Equivalence of binary strings and unary permutation DFAs

A DFA $\mathcal{A}$ is composite if its language $L(\mathcal{A})$ can be decomposed into an intersection $\cap_{i=1}^{k} L(\mathcal{A}_i)$ of languages of smaller DFAs. Otherwise, $\mathcal{A}$ is prime. This notion of *primality* was introduced by Kupferman and Mosheiff in [19], and they proved that we can decide whether a DFA is composite in ExpSpace and later in [20], the decomposition question for unary DFAs was proven to be in LogSpace. In the paper [21] by Jecker, Mazzocchi, and Wolf, they provided a LogSpace algorithm for commutative permutation DFAs, if the alphabet size is fixed. This also puts the bounded k-composite question for unary DFAs in the LogSpace complexity class, since the unary DFAs we obtain from parsing labels, are commutative permutation DFAs by definition. Refer to Table 2.1 for comprehensive details on complexity bounds. Typically, the decomposition of unary DFAs induced by its clean quotients. Consider a unary DFA represented as $A = \{\Sigma, Q, q_I, \sigma, F\}$. A clean quotient A_d of A is a DFA created by folding its cycle of length l to a cycle of length d, for some strict divisor d of l. Formally, A_d is induced by the equivalence relation $\sim_d$ defined by:

$$q_i \sim_d q_j \text{ if and only if } i \equiv j \ mod \ d$$

We obtain a DFA $A_d = \{\Sigma, Q', [q_I], \sigma', F'\}$, where Q' is the set of equivalence classes $[p]$ of the states $p \in Q$ The transition function σ' is defined such that for all $a \in \Sigma$ we have that $\sigma'([p], a) = [\sigma(p, a)]$, and F' is composed of the classes $[p]$ such that there is $q \in F$ such that $p \sim q$. F' is defined in a manner, that the language of A_d overestimates that of A, i.e., $L(A) \subseteq L(A_d)$, as every word accepted by A implies acceptance by A_d.

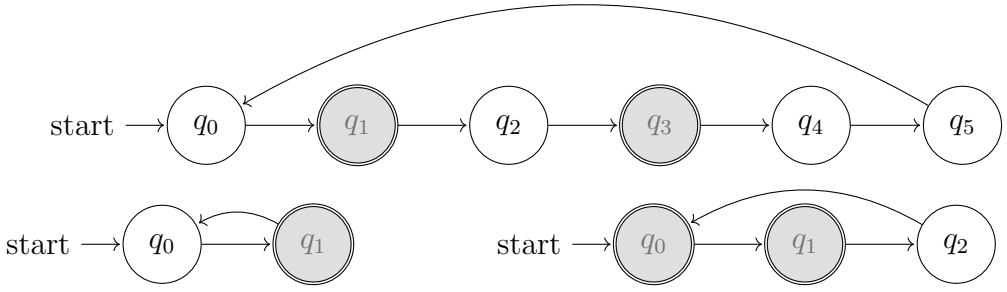


Figure 3.3: An example of a DFA A and its quotients A_2 and A_3

In the example in Figure 3.3, the DFA A with final states q_1 and q_3 can be replaced by the clean quotients A_2 and A_3 as there is no $q_i \in Q \backslash F$ in A where both clean

quotients A_d are in a final state $q_{i \bmod d}$. We can observe the over-approximation effect in the example depicted in Figure 3.3 as there is a state q_0 in A where the clean quotient A_2 is in a final state but A_3 is not and only the combination of both quotients is a valid decomposition of the original DFA. Finding quotients is trivial but the number of potential quotients rises sublinear with the number of states or the length of the edge label. Given a positive integer n, let $d(n)$ denote the number of divisors of n including 1 and n, so for example $d(8) = 4$. More generally, if n has a prime factorization $n = p_1^{a_1} \dots p_k^{a_k}$ then by the fundamental theorem of arithmetic $d(n) = (a_1 + 1) \cdot \dots \cdot (a_k + 1)$. Clearly, $d(n) \leq n$ and therefore $d(n) = n^{o(1)}$. This approach has to evaluate all divisors of l, which is not necessary to answer the DFA Decomposition question. By only considering maximal divisors the complexity can be further improved, which will be described in the next chapter.

3.2 Decomposition of DFAs Using Maximal Divisors

The idea of folding DFAs was greatly improved by Jecker, Mazzocchi, and Wolf in [21] by limiting the potential factors to maximal divisors instead of strict divisors of d. This can be done because every factor with length i_1 where i_1 is not a maximal divisor of $|Q|$, there is a factor with length i_2 and an integer $n \in \mathbb{N}$ such that $n \cdot i_1 = i_2$ and the factor with i_2 will also cover all the values of the i_1 factor. This property will be useful in another way later in Chapter 6 when searching for shorter factors instead of as few factors as possible.

Algorithm 3 from [21], solves the bounded decomposition problem for an unary DFA A, by iterating over all $2^{\mathcal{W}}$ possible word combinations from $\mathcal{W} = \{a^{|Q|/p_i} | 1 \geq i \geq m\}$ and finding a selection of k words from the set, that cover all the rejecting states of A. For each set of binary strings with the length of a maximal divisor of $|Q|$ and $\leq k$ ones, the word combination is tested by checking if there are any non-final states, not covered by the current set. Iterating over all states $q \in Q \backslash F$, the function *coverBySet* is called, to find states which are not covered by the current set. In the function, we go through all k words in the set and check if q is covered by the word $a^{|Q|/p_i}$ by calling the *cover* function from Algorithm 2 from [21]. Here $Q_{q,w_i} = \{\sigma(q, w_i^{\lambda}) | \lambda \geq |Q|\}$ is generated for the word $a^{|Q|/p_i}$. If the intersection of Q_{q,w_i} and the final states F of A is empty, $a^{|Q|/p_i}$ covers only rejecting states, and if this is true for all words from the current word combination set, a valid decomposition is found. The complexity of the given decomposition algorithm is LogSpace since the prime factor decomposition and the checking of the word combination is in LogSpace. The prime factor decomposition of $|Q|$ is in space logarithmic in $|Q|$ since A is given in unary. As the set of word combinations $\mathcal{W}$ only contains words of the length of a maximal divisor of $|Q|$, $|\mathcal{W}|$ is also logarithmic in $|Q|$ and iteration over all sets of $2^{\mathcal{W}}$ is also possible in LogSpace, by using a binary string which indicates the characteristic function.

For the AND-decomposition problem for DFAs this is as good as it gets, but since we actually want to decompose labels to increase understandability, we will focus on the OR-decomposition and finding decompositions with different properties than minimal width. We can still evaluate the approach using maximal divisors only as potential factor size while searching for an OR-decomposition of binary labels.

4. AND vs OR Decomposition

All theoretical studies so far used the AND-decomposition with the complement over all factors. This idea is easily expandable to the OR-decomposition, where the union over all factors B_i results in the original language, $L(A) = \cup_{1 \leq i \leq k} L(B_i)$. Note that, all B_i in the decomposition satisfy $|B_i| < |A|$ and $L(A) \supseteq L(B_i)$. With this change, a single factor implies certain final states, whereas, with the AND-decomposition, all factors have to be considered to know whether a given state is final in A. As we defined the OR-decomposition as a conceptual counterpart to the AND-decomposition, we considered different DFAs with either an AND-decomposition or an OR-decomposition. Of course, there exist DFAs with both and- and OR-decomposition but we also prove that this is not the case for every DFA. For an example see Figures 4.1 and 4.2, where A has a AND-decomposition and B has a OR-decomposition but not vice versa.

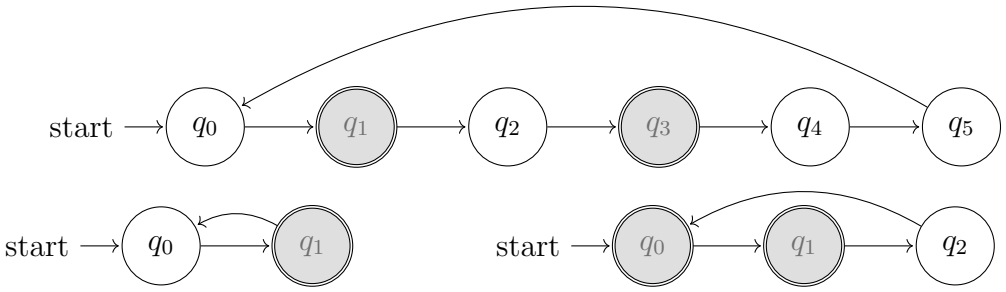


Figure 4.1: The DFA A and its factor A_2 and A_3 (AND-decomposition)

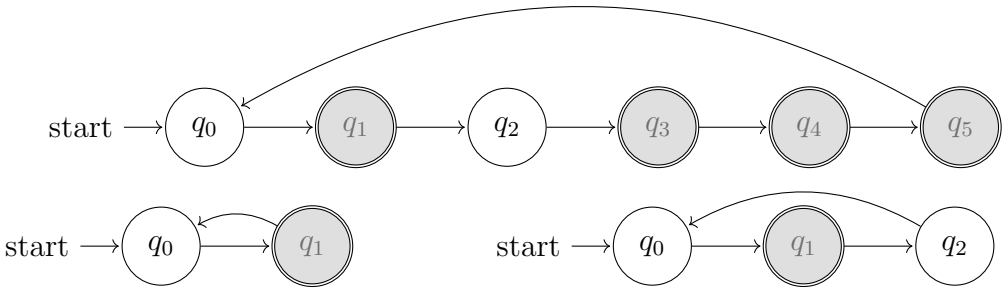


Figure 4.2: The DFA B and its factors B_2 & B_3 (OR-decomposition)

To prove that there exist words that are and-decomposable but not OR-decomposable and vice versa, we need a bit of background. For this purpose we have extended the logic operators $\vee$ and $\wedge$ to binary words. We also need the following Lemma:

Lemma 1. *Consider w, a binary string and A the corresponding unary* DFA*. Let the complement of w be w' where every value is flipped and A' its corresponding* DFA*. An AND-decomposition of A' is equivalent to an OR-decomposition of A.*

Proof. Since an AND-decomposition covers the non final states of a DFA and an OR-decomposition covers the final states, an AND-decomposition of the complement of a DFA will cover its final states, exactly like an OR-decomposition and vice versa. □

Theorem 1. *There exists a word $w \in \{0,1\}^*$ such that w is OR-decomposable but not AND-decomposable.*

Proof. Assume the DFA B from Figure 4.2. As from Lemma 7 from [20], for an AND-decomposition it suffices to consider the quotient DFAs of B. This implies that there exists no AND-decomposition for DFA B since 2 and 3 are the only divisors and the possible factors of this size, do not generate a valid AND-decomposition. □

Theorem 2. *There exists a word w such that w is AND-decomposable but not OR-decomposable.*

Proof. Let A be the DFA from Figure 4.1. Consider the complement DFA of A, A' with $F' = \{q_0, q_2, q_4, q_5\}$. Since the DFA has 6 states, it has only 2 potential

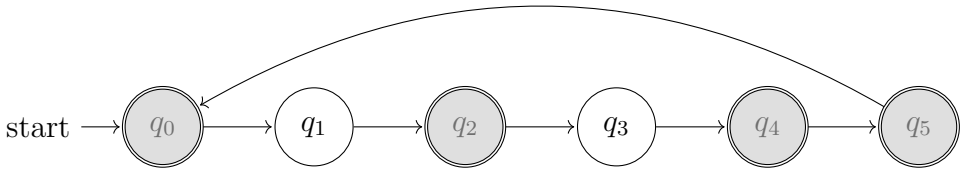


Figure 4.3: The DFA A', complement of A

quotients with size 2 and 3. The non final states q_1 and q_3 are neither periodic in the length 2 or 3, so both quotients are empty as they consist only of final states. With both quotients empty, there is no valid AND-decomposition of the DFA A' and therefore A does not have a valid OR-decomposition. □

Corollary 1. *The set of OR-decomposable strings and the set of AND-decomposable string is not comparable.*

As we have now proven that the results of AND-decomposition and OR-decomposition are not comparable, we will will focus on OR-decomposition for explainability purposes, but still evaluate the effectiveness of both methods.

5. Explainability

Understanding and interpreting labels in edge periodic temporal graphs is essential for various applications, such as network analysis, system monitoring, and predictive modeling. However, assessing the ease with which humans can comprehend these labels presents a unique challenge. In our earlier exploration of a simplified train schedule in Section 1.1, we handled a small scenario with short factors that are easy to comprehend. Handling real-world data with thousands of edges with labels with thousand time-steps, this simplicity of looking at a factor and guessing its semantic meaning gets lost. In this context, we propose a comprehensive metric to measure the "explainability" of decompositions of such labels, taking into account multiple effects that influence their possibility to interpret. This metric aims to enhance our ability to analyze a decomposition and comprehend the underlying factors, contributing to a more robust and insightful analysis.

5.1 Explainability Metric

To get some insight, consider the following imaginary example decompositions $\mathcal{D}$ to $\mathcal{D}^4$ of a DFA with $|Q| = 100$ and $|F| = 50$. The decomposition $\mathcal{D}$ for example, consists of the factors $\mathcal{D}_4$ and $\mathcal{D}_5$. The factor $\mathcal{D}_4$ has 25 outliers, the combined factors up to $\mathcal{D}_5$ have zero outliers. A zero in Figure 5.1 for any given factor size, implies that the decomposition with largest factor $\mathcal{D}_i$ of that size has zero outliers and covers the complete input DFA with a factor of this size, a dash implies that a decomposition with largest factor of this size covers values at all. For a DFA A,

	outliers of decomposition with largest factor $\mathcal{D}_i$ of size				
	4	5	10	20	50
$\mathcal{D}$	25	0			
$\mathcal{D}^1$	-	30	20	0	
$\mathcal{D}^2$	-	-	-	40	2
$\mathcal{D}^3$	-	-	-	45	29
$\mathcal{D}^4$	-	-	-	-	40

Table 5.1: Example decompositions and their outliers of combined factors

and its decomposition $(B_i)_{1 \leq i \leq k}$, we evaluate the following metrics:

Decomposition Structure ($\mathrm{DS} = \prod_{factor\ B_i} \frac{|B_i|}{|\mathrm{A}|} \cdot (1 + \frac{out(\{B_1,\ldots,B_i\})}{|F|})$)

The decomposition structure is the sum of all relative factor sizes, weighted by their relative amount of outliers. It takes into account whether the label is presented as

a single, continuous binary string or is split into multiple, shorter labels. Labels with multiple shorter segments may facilitate explanation, as they enable a more granular understanding of connectivity changes at different time steps if they cover a relatively high amount of values. For example, dividing the label into daily or weekly segments that represent more or less the long temporal label could aid in interpreting temporal patterns. A few small factors with few outliers imply a small decomposition structure value, which is visible in Table 5.2 for decomposition $\mathcal{D}$, a large factor, covering not many values implies a high decomposition structure value, see decomposition $\mathcal{D}^4$. Generally, a low DS value is expected to be more explainable by a human interpreter. Initially it was defined as:

$$DS_{rev1} = \sum_{factor\ B_i} \frac{|B_i|}{|\mathrm{A}|} \cdot \frac{out(\{B_1, \ldots, B_i\})}{|F|}$$

but that ignored the larges factor if it covered all values, as it was weighted by zero. This lead to decompositions with a small factor covering all values having the same decomposition structure metric as a decomposition with a single large factor covering all values. Also decompositions with a single factor with many outliers is ranked better, than a decomposition with two factors and less outliers, as visible in Table 5.2, comparing $\mathcal{D}^3$ and $\mathcal{D}^4$ in DS_{rev1} So the weight was set to be in range $[1, 2)$ by setting the definition to:

$$DS_{rev2} = \sum_{factor\ B_i} \frac{|B_i|}{|\mathrm{A}|} \cdot (1 + \frac{out(\{B_1, \ldots, B_i\})}{|F|})$$

This resolved the issue of the uncounted largest factor but another problem arose. As we were computing the sum, a decomposition with only one large factor and many outliers, was ranked better than a decomposition with many factors each covering only a few values. This is also visible in the last two decompositions in Table 5.2 in DS_{rev2}. Best results were archived with the current definition of the decomposition structure, where all imaginary decomposition in decreasing quality also gave a decreasing decomposition structure value.

Precision (precision $= \frac{|F| - out(\{B_1, \ldots, B_k\})}{|F|}$)

Precision refers to the regularity of patterns in the label across different time steps. A decomposition that has a high precision value, implies that there are few outliers and many of the actual values can be read from the factors. Low precision implies that the decomposition covers only a few values for example in decomposition $\mathcal{D}^4$ in Table 5.2, requiring either a list of all outliers or a decomposition with largest factor as large as the original DFA, to fully understand the full picture. A high precision is always preferred for an explainable label.

Size (k)

The decomposition size is fundamental in assessing explainability. A collection of labels with distinct information is generally easier for humans to understand, as it conveys information succinctly. Therefore, generally, a decomposition with many factors is preferred over a decomposition with few factors.

In this example, $\mathcal{D}$ is considered to be a very good decomposition after all metrics. It has only two factors of size 4 and 5, that cover the original DFA with a precision of 100%. The decomposition structure is small as the two factors are small relative

	outliers of decomposition with largest factor $\mathcal{D}_i$ of size					DS_{rev1}	DS_{rev2}	DS	precision in %	size
	4	5	10	20	50					
$\mathcal{D}$	25	0				0.02	0.11	0.003	100	2
$\mathcal{D}^1$	-	30	20	0		0.07	0.22	0.011	100	3
$\mathcal{D}^2$	-	-	-	40	2	0.18	0.88	0.187	96	2
$\mathcal{D}^3$	-	-	-	45	29	0.47	1.13	0.3	26	2
$\mathcal{D}^4$	-	-	-	-	40	0.40	0.90	0.9	20	1

Table 5.2: Example decompositions and metrics

to the input size and the second factor is only included in the sum with a weight of 1 if there are no outliers. $\mathcal{D}^1$ is similarly well decomposed, with a small decomposition structure and also a precision of 100%. Although it has slightly larger factors, there might be a benefit in having more factors with different sizes when extracting semantic meaning. $\mathcal{D}^2$ only consists of only two larger factors, but has a few hard outliers, compared to $\mathcal{D}^3$ with a precision of 26%. The last decomposition $\mathcal{D}^4$ has only a large factor with many hard outliers, which implies that it is hard to explain in a periodic way. In general, a low decomposition structure as well as high precision and size, allow for greater "explainability".

5.2 Explainability Measurement

To measure the proposed explainability metric for edge periodic temporal graphs, a systematic approach is employed, utilizing real-world data from face-to-face graphs. Existing algorithms for DFA AND-decomposition as well as novel approaches designed for explainable OR-decomposition construction are used and the resulting decompositions are evaluated using the described metrics. For a more complete analysis of the quality of the measured explainability, a study with a human-computer interaction focus would be required but is out of scope for this project. Motivating the example from the introduction, an experiment quantifying the possibility with which humans can comprehend the structure of temporal or edge periodic graphs could include a train network. If two groups of people are given the same network, one group becomes a description in the form of a temporal graph and the other an edge periodic multi-graph. Participants can then be asked questions regarding the traversal of the train network, or connectivity at certain times or intervals. By measuring correctness and required time of the given answers, it should be possible to evaluate, which representation is easier to understand. Alternatively, a qualitative analysis would be possible by performing one-on-one interviews, where similar tasks are given but the explainability could be measured by letting participants describe, how complex understanding one or the other data structure feels.

6. Novel Approaches

We now have a concept of what an "explainable" edge label looks like, so we propose new ideas for decomposing DFAs such that the resulting OR-decomposition has a good structure, precision, and size, resulting in good labels, understandable for humans. Additionally, we want the algorithm to be able to handle outliers of a given OR-decomposition, which might result in even shorter labels. If we have a OR-decomposition where a small factor covers most of the original DFA, but not all of it, we need either a larger factor or remember the uncovered values, which could increase explainability.

6.1 Greedy Short Factors

Searching for a decomposition using only the maximal divisors of $|Q|$, provides good solutions in LogSpace, but there are certain limitations. Consider for example Figure 6.1 depicting the DFA A with its factors A_2 and A_4. Since for this DFA, $|Q| = 8$, the only maximal divisor is 4 so the factor A_4 will be found by the original algorithm. In this particular example, there exists a smaller factor with size 2, A_2 which will only be found if all factors of $|Q|$ are checked instead of only the maximal divisors.

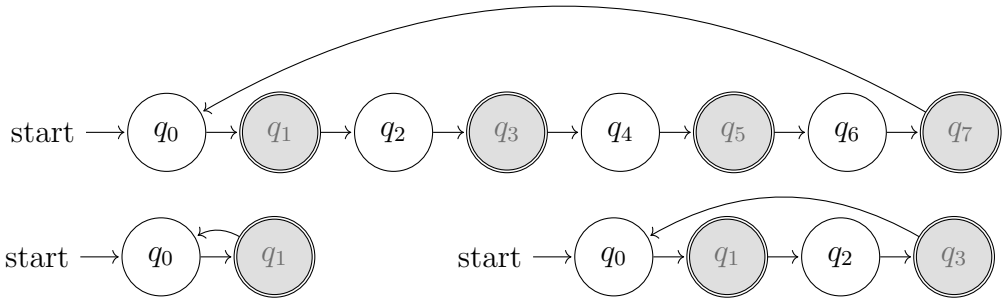


Figure 6.1: The DFA A and its factors A_2 & A_4

Usually, the algorithms and proofs consider unary DFAs consisting of a chain leading into a cycle of states. Since we obtain our DFAs by transforming from a periodic label, we only have DFAs with empty chains, therefore only considering unary permutation DFA. This allows us to use a slightly simplified version of the Algorithm 3 from [21] as visible in our Algorithm 1 as we do not encounter unary automata with $\sigma(q, uv) \neq \sigma(q, vu)$, and we want to compute an OR-decomposition. Instead of iterating over all potential sets of words $2^{\mathcal{W}}$ from $\mathcal{W} = \{a^{|Q|/p_i} | 1 \geq i \geq m\}$, we iterate over divisors of $|Q|$ and generate possible factors. For each divisor d we call the method *getFactor* and generate the potential factor A_d, where Q' is the set of equivalence classes $[p]$ of the states $p \in Q$, σ' is defined as $a \in \Sigma$ we have that

$\sigma'([p], a) = [\sigma(p, a)]$. Iterating over all states of the factor in the method *allAccepting*, we check if we are in a finial state and if all states folded onto the state $\sigma(q_I, a^i)$ are also final in the original DFA. If this is the case, we include it in the final states of the current factor. We are not only interested in answering the yes/no question of the composite problem but we want to collect the factors and continue our computation. After all possible factors are computed, they are added greedily to the decomposition, in the order of their length, until no further changes occur. This algorithm finds more factors than required since it also finds the divisors of the

Algorithm 1: Algorithm for greedily computing an OR-decomposition for unary DFAs and returning the factors.

```
Function findGreedyComposite(A = {a, Q, q_I, σ, F}: unary DFA):
    Decomposition ← ∅
    foreach A' ∈ getAllFactors(A) do
        if factorChangesComposite(A, A') then
            Decomposition.add(A')
    return Decomposition
Function getAllFactors(A = {a, Q, q_I, σ, F}: unary DFA):
    FactorList ← ∅
    foreach divisor d of |Q| do
        FactorList.add(getFactor(A,d))
    return FactorList
Function getFactor(A = {a, Q, q_I, σ, F}: unary DFA, integer d):
    F' ← ∅
    A_d ← {a, Q', [q_d], σ', F'}
    foreach i ∈ [1, d] do
        if σ(q_I, a^i) ∈ F and allAccepting(A, i, d) then
            F' ← F' ∪ σ(q_I, a^i)
    return A_d
Function allAccepting(A = {a, Q, q_I, σ, F}: unary DFA, integer d, integer i):
    pos ← (i + d) mod |Q|
    while pos ≠ i do
        if σ(q_I, a^pos) ∉ F then
            return false
        pos ← (pos + d) mod |Q|
    return true
```

maximal divisors. In contrast, not all found factors are required to form the decomposition and we only want to add factors, which change the decomposition. In our example in Figure 6.1 we find A_2 and A_4, but A_2 is fully sufficient for decomposing the DFA A. By iterating over all possible factors ordered by size, we can greedily add factors, that cover accepting states that have not yet been covered. The details of the method *factorChangesComposite*(A, A') are omitted, as it simply checks if the addition of the factor produces a decomposition with fewer outliers than currently present. To address the problem that most of the labels do not have proper decompositions in the mathematical sense, we will also consider a set of smaller automata

that do not cover every value of a decomposition. For this, we want to keep track of the outliers of each factor of the decomposition and the outliers of decomposition itself. Regarding explainability, it should be of preference, to have a small factor but a couple of outliers, in comparison to a long factor, and even have a decomposition with hard outliers, e.g. values that cannot be covered by a decomposition. In this case, a decomposition including outliers is also preferred.

6.2 Problem Generalization

The original problem was solvable in LogSpace, but by extending the problem to finding the shortest possible factors, we face a different challenge. Finding a valid decomposition is possible in LogSpace but selecting a minimal amount of given factors needed to be a valid decomposition is not. This problem can be generalized for picking a minimal decomposition from a given set of factors as follows:

Definition 6.2.1 (Select Factors Decomposition)**.**
Input: A set of binary strings $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ *with* $\ell(w_i) < l$ *and* $\ell(w_i)$ *divides* l *for* $1 \leq i \leq n$*, a binary string* u *with* $\ell(u) = l$ *and an integer* k*.*
Question: Exists indices $i_1, i_2, \ldots i_k$ *such that*

$$w_{i_1}^{l/\ell(w_{i_1})} \vee w_{i_2}^{l/\ell(w_{i_2})} \ \ldots \ \vee w_{i_k}^{l/\ell(w_{i_k})} = u$$

To find lower bounds for this novel problem, we are going to reduce the Set Cover problem, known to be NP-complete, to our novel Select Factors Decomposition problem, using a polynomial time reduction.

Definition 6.2.2 (Set Cover)**.**
Input: A set of elements $U = \{u_1, u_2, \ldots, u_n\}$*, a set* $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ *where* $C_i \subseteq U \ \forall \ C_i \in \mathcal{C}$ *and an integer* k*.*
Question: Exists a set $\mathcal{O} \subseteq \mathcal{C}$ *of size at most* k *such that* $\bigcup_{C_i \in \mathcal{O}} C_i = U$*.*

To reduce the Set Cover problem to our novel problem, we create an instance as follows. For the universe $U = \{u_1, u_2, \ldots, u_n\}$ we will construct a binary string $u = 1^{2|U|}$ with length $\ell(u) = 2|U|$. For each subset of $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ we create a word w_i such that $w_i[j] = 1$ iff $u_j \in C_i$.

Assume a solution to the Select Factors Decomposition problem $i_1, i_2, \ldots, i_k$ that chooses from the words w_i to form u, then the respective subsets $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$ in the constructed Set Cover instance cover the entire universe U. Assume there exists an element $u_x \in U$ with $u_x \notin C_{i_k} \ \forall \ i_1, i_2, \ldots, i_k$. This implies that there is no $w_{i_k} \ \forall \ i_1, i_2, \ldots, i_k$ with $w_{i_k}[x] = 1$ which is in contradiction to the found solution with $w_{i_1}^{l/\ell(w_{i_1})} \vee w_{i_2}^{l/\ell(w_{i_2})} \ \ldots \ \vee w_{i_k}^{l/\ell(w_{i_k})} = u = 1^{2|U|}$. This means that u_x cannot exist and we found a Set Cover of size k. If the Set Cover problem has a solution $\mathcal{O} = \{C_{i_1}, C_{i_2}, \ldots, C_{i_k}\}$ with $\bigcup_{C_i \in \mathcal{O}} C_i = U$, then $w_{i_1}^{l/\ell(w_{i_1})} \vee w_{i_2}^{l/\ell(w_{i_2})} \ \ldots \ \vee w_{i_k}^{l/\ell(w_{i_k})}$ is a solution to the Select Factors Decomposition problem. Assume $w_{i_1}^{l/\ell(w_{i_1})} \vee w_{i_2}^{l/\ell(w_{i_2})} \ \ldots \ \vee w_{i_k}^{l/\ell(w_{i_k})} \neq u$, that implies there is at least one position x where $(w_{i_1}^{l/\ell(w_{i_1})} \vee w_{i_2}^{l/\ell(w_{i_2})} \ \ldots \ \vee w_{i_k}^{l/\ell(w_{i_k})})[x] \neq u[x]$. This is in contradiction to the construction as there has to be a $C_{i_x} \in \mathcal{O}$ with $u_x \in C_{i_x}$ and therefore a word w_{i_x} with $w_{i_x}[x] = 1$. This reduction demonstrates that solving the Select Factors Decomposition problem is as hard as solving the Set Cover problem.

6.3 Fourier Transform

Finding shorter factors than just the maximal divisors should already increase the explainability but consider the following example from Figure 6.2. Factor A_4 alone would suffice to cover the original DFA A but most of the final states are already covered by the smaller factor A_2. Consider the states q_1, q_3q_5 and q_7 which are covered by both factors, which is not required for the decomposition to be valid. We can remove all the states from A_4 which are already covered by A_2 and this results in 2 Factors with each having just one final state. Of course, there might be multiple states which are not part of any other smaller factors, but any factor of an OR-decomposition can be replaced by multiple factors of the same length, each with $|F_{B_i}| = 1 \; \forall \; (B_i)_{1 \leq i \leq k}$. Each factor only having a single final state further increases explainability as it allows for each value to be represented as the position and the size of the factor or the periodicity, e.g. instead of a binary label, a list of integer tuples could represent the label. For a comparison between single or multiple binary labels and a list of tuples with the factors, a detailed examination including human-computer interaction would be required.

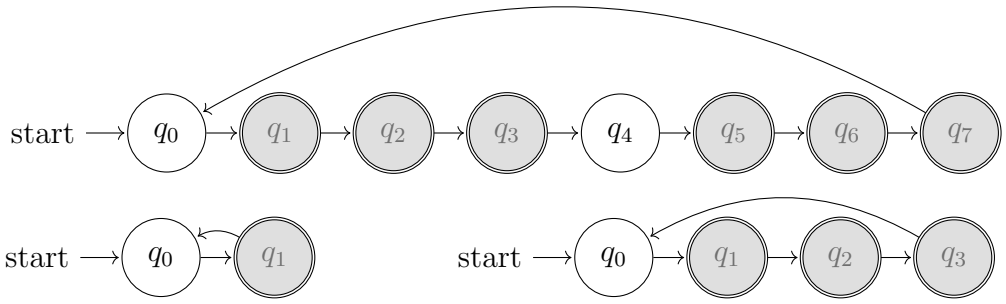


Figure 6.2: The DFA A and its overlapping factors A_2 & A_4

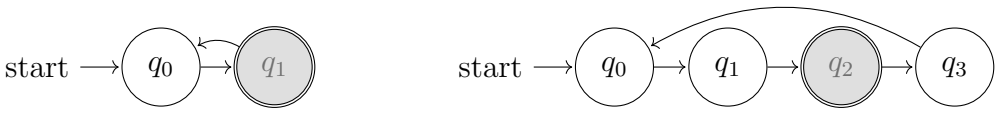


Figure 6.3: The Fourier transformed factors A_2 & A_4

Of course, this would also greatly increase the amount of factors required for a valid decomposition, and in turn, could hurt performance. In general, the greedy approach of combining factors of increasing size should still suffice.

6.4 Relaxation of Requirements

So far we only considered decompositions where the union over all factors B_i results in exactly the original language, $L(A) = \cup_{1 \leq i \leq k} L(B_i)$, and all factors B_i in the decomposition satisfy $|B_i| < |A|$. Since our goal is not to find decompositions of minimal width but to find decomposition such that the explainability is maximized, we can relax the requirements to achieve greater explainability. If we allow for factors of original size $|B_i| \leq |A|$, we can find a factor of the original size, which covers all values not covered by a factor so far. The values only covered by this final factor B_k but not by any other factor, can be considered hard outliers and

it allows for a decomposition of any DFA. This might not make much sense for DFAs but it makes sense in the context of explainable labels. Additionally, we may allow for $L(A) \supseteq \cup_{1 \leq i \leq k} L(B_i)$ and use the precision for a decomposition with $L(A) \neq \cup_{1 \leq i \leq k} L(B_i)$ as $precision(\cup_{1 \leq i \leq k}\ B_i) = \frac{|F_A| - out(\{B_1 \ldots B_k\})}{|F_A|}$ used in the metric, and allow for a decomposition to be valid if a certain precision is reached.

6.5 Delta Window as Preprocessing or During Folding

Working with real-world data always poses a challenge as the dataset might be very large or contain measurement errors. The longer a label and therefore the corresponding automata gets, it becomes less and less likely that a valid folding of the automata exists. To mitigate this problem, we implemented a delta window during the folding step of the algorithm. The delta window during folding does not change the input, but instead, a folding is considered valid, if a set value folds onto another by some delta window. Formally, with applied delta window of width w, a state q_i is final in the input, if there exists a $q_j \in F$ with $j \in \{((i - w)\ mod\ |Q|), \ldots, ((i + w)\ mod\ |Q|)\}$. The effectiveness of the delta window approach will be evaluated in Section 8.3.

6.6 EPG Creation from Decomposition

In the last step, our goal is to create an EPG instance by combining decompositions across all labels. For each edge and its corresponding label, we gather all factors from the computed OR-decomposition and substitute them with multiple edges, each representing a factor. Ideally, the resulting edge labels should be smaller than the original label. In cases of severe outliers, an alternative representation must be introduced. This can take the form of the original label, serving as a fallback for outliers, or as a list where an edge is included whenever the current time step $t\ mod$ the original label length equals an element in the list. While this method becomes less efficient with numerous hard outliers, it could still be an improvement compared to using a lengthy label.

7. Implementation

All algorithms described above have been implemented using Kotlin because it can be compiled for the Java Virtual Machine or native execution. The project is realized as a maven project, to simplify dependency management. Although all theoretic results have been performed for DFAs, since none of the properties of a DFA is actually used except for the set of final states, the input as well as the factors are represented as boolean arrays, where a 1 indicates a final state. This benefits performance, as array operations are usually faster than e.g. lists of custom objects and this also enabled easy transformation between labels and boolean arrays and vise versa.

7.1 File Reader

Initially, a network from the F2F set described in Section 2.3 is selected and the file reader parses the desired network id and the number of participants from the `network_list.csv` file. With these metadata, the selected dynamic face-to-face interaction network is parsed from the `network/network[ID].csv`. Each file has the time series of the who-looks-at-whom networks where each row consists of a timestamp and a binary adjacency matrix. The binary variables in the adjacency matrix indicate where participant i is looking at in the current timestamp, either one of the other participants or the laptop in front. As a participant can look at only one direction at a given timestamp, only one of the binary variables is 1 per row. The file reader parses the adjacency matrix and returns a JGraph object with boolean arrays to represent the edge labels. For each label then a decomposition is performed, using the specified algorithm.

7.2 Decomposition using Maximal Divisors

For the initial implementation, only maximal divisors were used as this is already well studied. For an input array of size n, all maximal divisors are lazily yielded into a sequence, via a *NumbersExtensions*. Extension functions are a useful Kotlin tools if you want to extend the static functionality of base classes, e.g. Integers. Then a preliminary Factor is created for each maximal divisor, with an empty array the size of the divisor, and a job is spawned for each factor as the periodicity of the factors can be tested in parallel. With the help of Kotlin co-routines, a modest multi-threading approach is implemented where the main algorithm thread waits for all the factors to be checked. Afterward, they are collected into a *Cover* object, which handles all the validation and quality assessment. For this, a cover array is created, where all factors are consolidated and uncovered values, so-called outliers, are visible. As not all inputs are decomposable and by default, we do not allow

$|B_i| = |A|$, we may find labels for which the DFAs are not decomposable. In this case, we consider them partially decomposable, as the found factors create a cover array that is partially identical to the original but not in all places, so a warning is issued with the number of hard outliers.

7.3 Greedy Short Factor Decomposition

The implementation of the greedy approach to identify shorter factors closely resembles the previous method. As explained in Section 6.1, we consider all divisors of the length of the original DFA instead of just the maximal divisors. Multi-threading in the job implementation becomes even more advantageous when dealing with a larger number of potential factors. All identified factors are once again gathered into a Cover object. However, this time, we only include factors in the cover object if their addition alters the resulting cover array. For instance, in the example illustrated in Figure 6.1, we would add A_2 and skip over A_4 since the further addition of factors doesn't alter the resulting decomposition. If, following the addition of a factor, a complete decomposition is formed with $L(A) = \cup_{B_i \in \text{Cover}} L(B_i)$, it is then validated by counting the outliers and checking for $out(B_i \in \text{ Cover}) = 0$. Given that a substantial number of DFAs generated from labels in real-world data are likely prime, we've implemented a decomposition with a specified $precision(B_i \in \text{ Cover})$, as well as a decomposition that allows for potential factors to be as large as the original DFA size.

7.4 Fourier Transformed Decomposition

With the goal of increased explainability, the Fourier-transformed decomposition is implemented according to Section 6.3. Similar to the the greedy approach, we consider all divisors of the length of the original DFA. Employing the Fourier transformation on potential factors involves an additional step where values of a factor B_i, that are also covered by a smaller factor B_j with $j|i$, are removed from B_i. Subsequently, we enforce the requirement for a single set value, $|F_{B_i}| = 1$ by replacing factors that fall short of this criterion with multiple factors that each satisfy it and together cover the same values. The aggregation into a cover is once again executed in a greedy fashion, aiming to minimize the number of factors in the cover for improved clarity. Regardless of the decomposition method employed, a *Cover* object can be transformed into an EPG multi-graph. This new graph retains the same nodes as the original F2F graph, but for each cover, there exists a multi-edge, corresponding to each decomposition with the cover array serving as the label.

7.5 External Libraries

Some external libraries are used throughout this project which was quite simple through the use of maven. The library *org.junit.jupiter* provided the packages *junit*, *engine* and *api* in version 4.13.2, which allows a simple test unit creation and execution. Multi-threading was enabled by the use of Kotlin *coroutines* in version 1.7.1, provided by JetBrains. JGraph [24], a Java library of graph theory data structures and algorithms, was used in version 1.5.1 for general data structures such as simple

and multi-graphs. Lets-Plot[1] is an open-source plotting library for statistical data and was used for analysis, debugging and visualizations used in this document. An argument parser[2] for Kotlin was used to facilitate the creation of a command line interface.

7.6 Usage

To enable a convenient usage, the algorithm is obtainable as jar file but it can also be built from sources. It provides a simple command line interface (CLI) for the EPG generator, providing functionality for generating EPG instances based on the F2F dataset and specified options. The basic usage involves selecting the temporal network id in the range of 0 to 61 and using the preset default values.

```
Usage: EPG Generator options_list
Arguments:
input -> input (Temporal Network id in range (0..61) { Int }
threshold [1.0] -> Min threshold of cover to be valid (optional) { Double }
Options:
--dotenv, -env [false] -> Use config from .env file
(Recommended usage due to amount of args)
--Mode of decomposing DFAs [GREEDY_SHORT_FACTORS] -> Choose how
a decomposition is found, using only maximal divisors,
using all factors and greedily collect up to the threshold
or perform a fourier transform for increased understandability.
{ Value should be one of [max_divisors, greedy_short_factors,
  fourier_transform] }
--Mode of composing factors [OR] -> Choose how a composition is formed,
using AND or OR operator for adding factors together to
a decomposition. { Value should be one of [and, or] }
--skipSelfEdges, -skipSelfEdges [false] -> Skip loop back edges
with same source and target, these are often useless.
--debug, -d [false] -> Turn on debug mode
--quiet, -q [false] -> Turn on quiet mode
--deltaWindowPreprocessing [0] -> Delta window
 in preprocessing of the label { Int }
--deltaWindowAlgo [0] -> Delta window during decomposing { Int }
--help, -h -> Usage info
```

As it comes with a lot of different modes and parameters, a `.env` file is also provided, for persisting arguments. The project is available on Github[3] and released with an MIT License.

[1] https://github.com/JetBrains/lets-plot
[2] https://github.com/xenomachina/kotlin-argparser
[3] https://github.com/fierg/ept-graph

8. Evaluation

In this evaluation section, we explore how well the algorithm performs by looking at the performance and examining the decompositions. Our primary interest lies in two aspects, the size of the factors contributing to the decomposition and the coverage of final states by increasing decompositions. Through this analysis, we aim to gauge the algorithm's effectiveness in generating meaningful decompositions. This evaluation is pivotal for understanding the algorithm's capability to produce concise and relevant factors, providing insights into the temporal structure of the given labels. For this purpose, the complete F2F dataset was evaluated, described in Section 2.3 consisting of 62 Temporal graphs with 20 to 56 edges each, ignoring loop edges. In total, there are 3321 edges with a combined label length of around 2.3 million giving an average label length of 6993. Please note that the data sets represent people looking at each other, a directed edge from node u to v indicates participant u looks at participant v therefore only one edge label at a given time step t has its value set, $\tau(e)[t] = 1$ for exactly one outgoing edged of u. This is also visible in the data as only 13.4% of values in the labels is set to 1 over all the labels.

8.1 Performance Evaluation

Although performance was not the main focus of the implementation and evaluation, generating OR-decomposition for the complete dataset is reasonably fast with around 3 seconds if using the maximal divisors or the greedy approach. The Fourier-transformation takes a lot of additional time for cleaning multiples of a factor as well as replacing factors with multiple set values with a set of factors with only one set value, causing it to run for 1 minute and 45 seconds. All benchmarks were performed on an AMD Ryzen 5 2600X six-core processor (12 threads) with a 3.6 GHz base clock and a 4.2 GHz boost clock speed. For memory, 16GB 3200MHz RAM and a Samsung EVO SSD was used for persistent storage. It is to be noted

	MaxDivisors	GreedyShortFactors	FourierTransform
OR-Decomposition	3.12	3.47	105
AND-Decomposition	9.85	24.83	-

Table 8.1: Decomposition time in seconds [s] for complete dataset

that the AND-decomposition is expected to be slower because of the considerably greater amount of zeros in the data set as well as the modification to the cover finding algorithm. While with an OR-decomposition, each additional factor can only remove outliers, in an AND-decomposition an additional factor can also add new outliers. Because of that, the current outliers are recalculated, considering all factors currently in the decomposition. The Fourier-transform method again increases the

number of factors per cover, which hurts performance. Also, it is not implemented for AND-decomposition as it is not considered reasonable, since a human would still need to look at all factors since it is an AND-decomposition.

8.2 Decomposition Evaluation

Moving on to the decomposition evaluation, we validate each method separately and then compare them against each other. We also benchmark AND-decomposition and OR-decomposition. To get an insight into how well a particular decomposition method is performing, we decompose the whole data-set and then plot each decomposition. For each decomposition, we plot its factors by its relative size compared to the original DFA size and the relative amount of covered values of the partial decomposition up to this factor.

8.2.1 Maximal Divisor Decomposition Evaluation

Decomposing with the Maximal Divisor method implemented as described in Section 7.2 was used first for decomposing the complete dataset. Analyzing the resulting decompositions, for each decomposition of a DFA A we plot each factor B_i in increasing size. For B_i we plot its size relative to the original DFA $\frac{|B_i|}{|A|}$ with the precision of the partial decomposition up to that factor. The results are shown

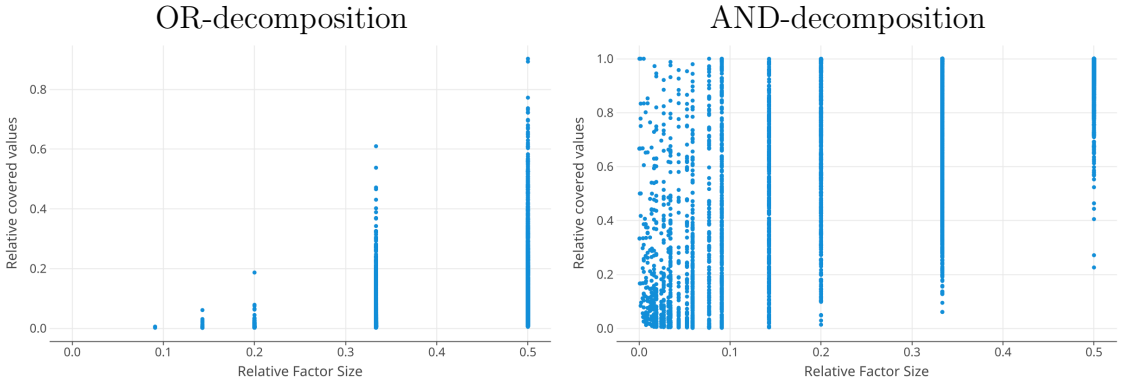


Figure 8.1: Relative amount of covered values by relative factor size

in Figure 8.1 and it clearly shows that the OR-decomposition finds decompositions, but there are fewer factors in the decompositions and they cover fewer values than the comparable AND-decomposition. A lot of data-points collapse onto a single x value since they are grouped by their relative size, not their absolute. For the AND-decomposition there are many values scattered around a small relative factor size. To get some more insight into the distribution, a box plot is also provided in Figure 8.2. Box plots visualize data using typically five values: the median, first quartile, third quartile, and the smallest and largest values within the lower and upper bounds of the distribution. The edges of the box are always the first and third quartile, and the line inside the box is the median. Lines extending from the boxes indicate the spread outside the upper and lower quartile and values outside ± 1.5 times the inter-quartile range are considered outliers of the distribution and will be depicted as individual points. Here, the same x and y values are chosen but the values are aggregated such that if a x value is too close to another, their data

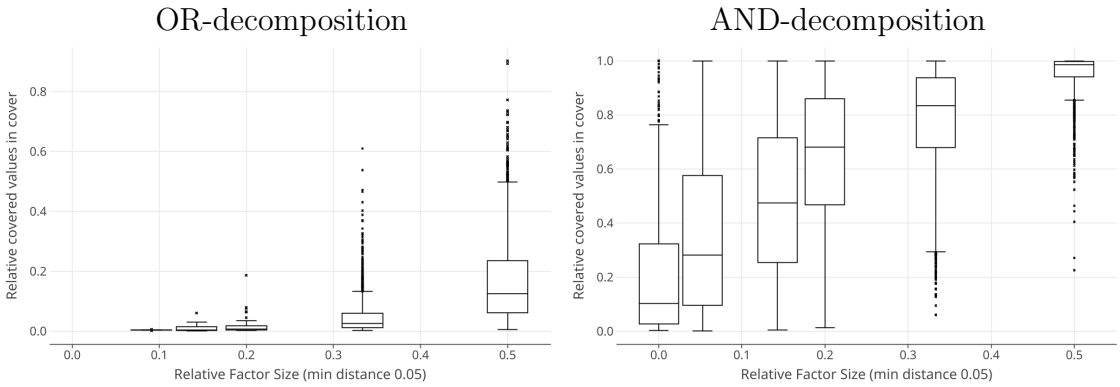


Figure 8.2: Relative amount of covered values by relative factor size (MaxDivisors)

points get merged. The OR-decomposition plot shows that even partial decompositions with largest factor size of $\frac{1}{2}$ which is the largest possible factor, the median over all these partial decompositions only covers 10% of the target DFA. There are some partial decompositions with largest factor $\frac{1}{3}$ or $\frac{1}{2}$ of the original size, which cover already up to 60-80% of the original DFA but they are outliers and most of the OR-decomposition found by using the maximal divisors cover almost nothing of the original DFA, especially partial decompositions with only small factors. For the AND-decomposition a lot of the small relative factor sizes have to be aggregated to able to visualize them as box-plots, so if a relative factor size value is less than $\frac{1}{20}$ away from a previous data-point, they become one distribution. Notably, partial decompositions with largest factors of size between $\frac{1}{12}$ and $\frac{1}{5}$ stretch over the entire range of relative covered values in the decomposition. With partial decompositions with largest factor of $\frac{1}{6}$ relative factor size, the AND-decomposition already covered more than 50% of a given DFAs final states.

8.2.2 Greedy Short Factor Decomposition Evaluation

Looking at the novel greedy approach for finding short factors, a similar picture becomes evident. We find partial decompositions with more factors in an OR-decomposition and some better ones as before when using just the maximal divisors as potential factor sizes, but some factor size data points from Figure 8.3 only have

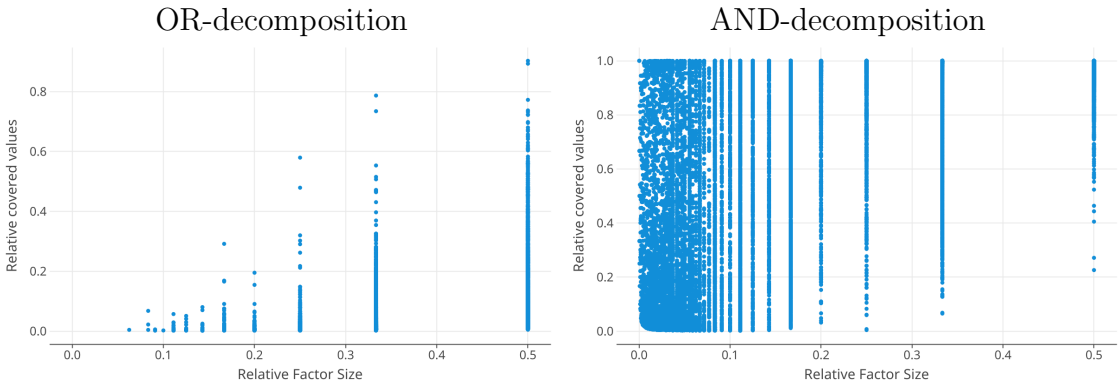


Figure 8.3: Relative amount of covered values by relative factor size

a single partial decomposition with a factor of that relative size. Although there are now more factors present per decomposition, the partial decompositions with

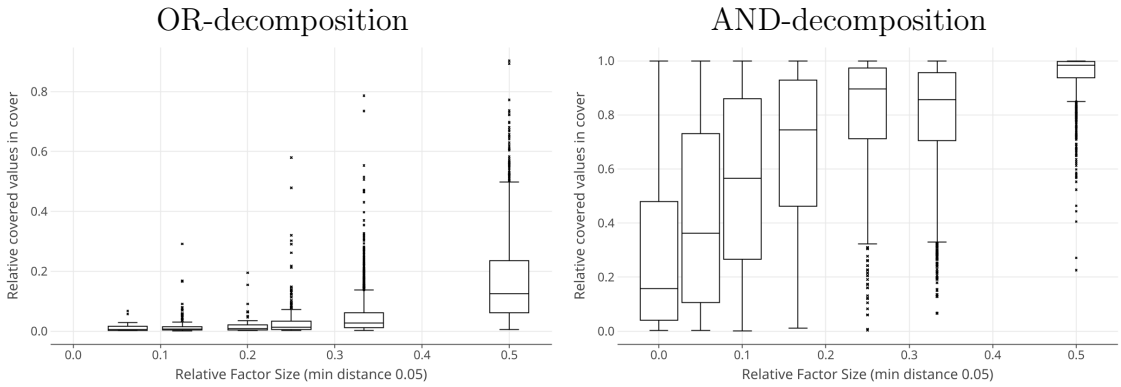


Figure 8.4: Relative amount of covered values by relative factor size (GreedyShortFactors)

short factors only cover at most 20% of the final states of the DFA they compose. As Figure 8.4 shows, we find decompositions with smaller factors and even some better outliers of the distribution, but the overall relative covered values by the decompositions stay very low. There are partial decompositions with factors of relative size $\frac{1}{3}$ or $\frac{1}{2}$ which cover up to 80% of the final states of their DFA present, but the median over the distribution never exceed 16%. This is partially expected, due to the real-world scenario and the way the data is generated. There are not that many 1 values in the labels, so the probability of them reoccurring after a periodic time is also low. It is also just not likely that a person looks at another person in an exactly periodic way, the data is temporal but just not very periodic in terms of 1 values. The AND-decomposition is very effective on the other hand, we find so many decompositions with small factors that the plot becomes a bit overwhelmed. Looking at the distribution of relative covered values, we see that even with partial decompositions of only 10% of the original size, the distribution is covering the complete range but the median is close to 60% of covered final states. This shows that the greedy approach of finding short factors, can find factors which cover more of the input at a smaller relative factor size. When enabling the delta window of width 1, the distributions change drastically, partial decompositions with factors of size $\leq \frac{1}{3}$ of the original DFA cover up to 100% of the input, with the median above 50%, see Figure 8.5 on the left. So the delta window drastically improves

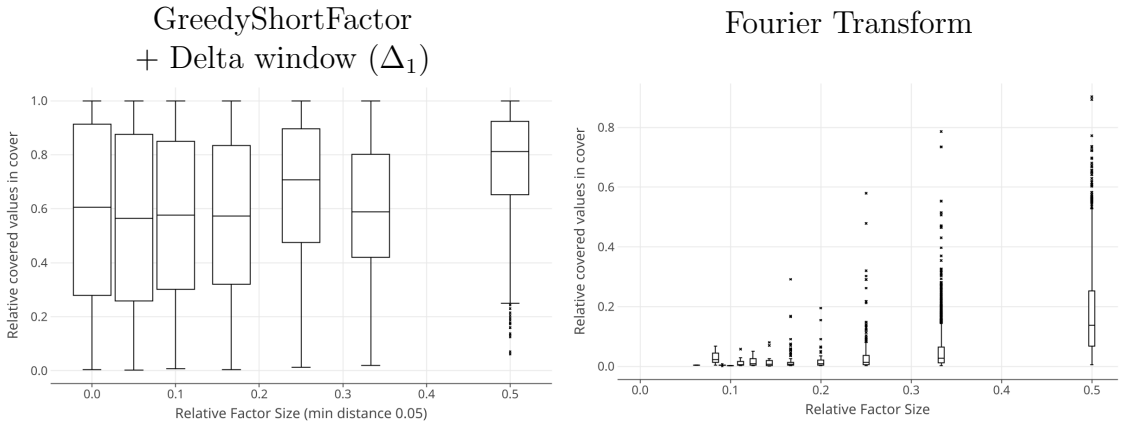


Figure 8.5: Relative amount of covered values by relative factor size (Fourier Transform & GreedyShortFactors with delta window of size 1)

the effectiveness of finding OR-decomposition when few values are set to 1, while on the other hand accuracy drops with each delta window increase. This has to be evaluated with a human computer interaction study to weight improve effectiveness against a washed out result, since a 1 in a label now implies that an edge is present in the range $[t - \Delta, t + \Delta]$.

8.2.3 Fourier-Transform Decomposition Evaluation

Moving on with the analysis of the Fourier-transformed factors, we see more factors in Figure 8.5 on the right, but with the same quality since we do not find new ones, we just separate them into their single state components. The goal for the Fourier-transformed OR-decomposition was the increase explainability, so we have to evaluate the metrics for this approach. For a proper analysis regarding the explainability and the possibility to comprehend the factors as periodic elements of a temporal graph, the effectiveness has to be evaluated in a qualitative way with human interaction.

8.3 Explainability Evaluation

Our analysis now turn towards the evaluation of the explainability metric, presented in Section 5.1. As our greedy methods do not find better decompositions, only more factors, the quality is the same for different methods. It is to be noted, that the precision over all graphs is very low for the general OR-decomposition, with only an average precision of 11.7% (see Figure 8.2) and only 3 valid decompositions.

Decomposition	Precision [%]		Decomposition Quality		
	avg.	median	valid	$p \geq 90$	$p \geq 75$
OR-decomposition	11.7	16.9	3	3	3
OR-decomposition + Δ_1	71.0	74.6	3	1091	1636
OR-decomposition + Δ_2	73.5	78.8	3	1277	1772
AND-decomposition	85.6	93.2	182	1908	2670

Table 8.2: Precision for OR-decomposition ($\cup$) and AND-decomposition ($\cap$)

When also enabling the delta window with width 1, the precision increases drastically and we have over a thousand decompositions with at least 90% precision. Further increasing the delta window to size 2 does improve precision, but not by that much. Even with the delta window size of 2 we do not encounter more valid decompositions. The AND-decomposition as on average much higher precision of 86% and finds 182 valid decompositions out of the 3321 edges, and 1908 decompositions with precision $\geq 90\%$. This is partially explained by the higher amount of 0 values in the input data as previously mentioned. Looking at the other metrics, decomposition structure and size of the decomposition, we see different result for different methods. The MaxDivisor approach finds very few factors per decomposition, on average,2.1 and the median is only 2 factors per decomposition when using an AND-decomposition and even less when using the OR-decomposition. The novel greedy approach finds more factors per decomposition and also smaller ones, subsequently the decomposition structure metric is better for AND-decomposition and OR-decomposition. When considering also the delta window of size one or two, we

Method		DS		Size		
		avg.	median	avg.	median	Σ
MaxDivisors	$\cup$	0.752	0.661	1.093	1	3629
	$\cap$	0.213	0.201	2.111	2	7010
GreedyShortFactors	$\cup$	0.718	0.661	1.233	1	4095
	$\cup + \Delta_1$	0.116	0.001	4.898	3	16265
	$\cup + \Delta_2$	0.115	0.001	5.281	3	17539
	$\cap$	0.125	0.006	5.193	3	17247
FourierTransform	$\cup$	0.066	0.001	2.219	2	7368

Table 8.3: Explainability metrics for different methods

find many more factors with up to 5.2 on average. With an decomposition structure value of 0.116 we find also small factors as part of the OR-decomposition, which is desired. The FourierTransform method is only viable for regular OR-decomposition but outperforms both other methods in terms of decomposition size and decomposition structure value. With an average decomposition structure of below 0.1 we seem to find decompositions with lots of small factors, increasing potential explainability.

9. Discussion

Moving on to the conclusion and discussion, the introduced methods for obtaining explainable factors for comprehensible edge labels works, although it is not particularly suited for the given real-world data-set, due to the abundance of zero values in the input labels. The greedy approach of finding shorter factors proved to be working and showed that an OR-decomposition can be found in some cases and that it ranked better in the explainability metric compared to the maximal divisor approach and had more factors in a decomposition. When accuracy is not of the greatest importance, a delta window approach can be deployed to further increase the effectiveness. The Fourier-transform approach of finding labels with only a single value set outperformed the decomposition structure metric but took considerably more time to compute. For the selected real word data-set, the OR-decomposition was outperformed by any AND-decomposition so it is also potentially interesting, to further analyze the possibility of visualizing AND-decomposition in a way, that humans can easily comprehend the provided information. As previously mentioned, these findings and the underlying metric should be validated in a human-computer interaction study to ensure the correctness and implications.

When extrapolating to the train schedule example, a train station does not necessarily have only one outgoing edge at a given time step as the face-to-face network does, but it is safe to assume that for a majority of time intervals, there is no train leaving a station. It might be applicable for a AND-decomposition or an OR-decomposition with flipped input, then a set value would imply no train leaving which is an unconventional but feasible way of providing information. Also, the Fourier-transformed factors allow for a different representation than a family factors each with multiple values set, as they just consist of the information at which index a periodic value sits, and with which size it periodically repeats, which was not visualized yet.

Bibliography

[1] Holme, P. and Saramäki, J., Temporal Networks, volume 519, pages 97–125, Elsevier BV, 2012.

[2] Erlebach, T., Hoffmann, M., and Kammer, F., On Temporal Graph Exploration, volume abs/1504.07976, 2015.

[3] Michail, O., An Introduction to Temporal Graphs: An Algorithmic Perspective, volume abs/1503.00278, 2015.

[4] Oettershagen, L. and Mutzel, P., TGLib: An Open-Source Library for Temporal Graph Analysis, 2022.

[5] Al-sharoa, E., Al-khassaweneh, M. A., and Aviyente, S., Detecting and Tracking Community Structure in Temporal Networks: A Low-Rank + Sparse Estimation Based Evolutionary Clustering Approach, volume 5, pages 723–738, 2019.

[6] Arrighi, E., Grüttemeier, N., Morawietz, N., Sommer, F., and Wolf, P., Multi-Parameter Analysis of Finding Minors and Subgraphs in Edge Periodic Temporal Graphs, 2022.

[7] Dunlavy, D. M., Kolda, T. G., and Acar, E., Temporal Link Prediction Using Matrix and Tensor Factorizations, volume 5, New York, NY, USA, 2011, Association for Computing Machinery.

[8] Cui, J., Zhang, Y.-Q., and Li, X., On the clustering coefficients of temporal networks and epidemic dynamics, pages 2299–2302, 2013.

[9] Fu, D., Fang, L., Maciejewski, R., Torvik, V. I., and He, J., Meta-Learned Metrics over Multi-Evolution Temporal Graphs, in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 367–377, New York, NY, USA, 2022, Association for Computing Machinery.

[10] Kerracher, N., Kennedy, J., and Chalmers, K., The design space of temporal graph visualisation, 2014.

[11] Orlando, D., Ormachea, J., Soliani, V., and Vaisman, A., TGV: A Visualization Tool for Temporal Property Graph Databases, 2023.

[12] Dorner, M., Smite, D., Méndez Fernández, D., Wnuk, K., and Czerwonka, J., Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs, pages 195–204, 2022.

[13] Tang, J., Musolesi, M., Mascolo, C., and Latora, V., Temporal distance metrics for social network analysis, in *Proceedings of the 2nd ACM workshop on Online social networks*, pages 31–36, 2009.

[14] Fu, D. and He, J., DPPIN: A biological repository of dynamic protein-protein interaction network data, in *2022 IEEE International Conference on Big Data (Big Data)*, pages 5269–5277, IEEE, 2022.

[15] DiBrita, N. S., Eledlebi, K., Hildmann, H., Culley, L., and Isakovic, A. F., Temporal graphs and temporal network characteristics for bio-inspired networks during optimization, volume 12, page 1315, MDPI, 2022.

[16] Qin, H. et al., Mining Periodic Cliques in Temporal Networks, in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1130–1141, 2019.

[17] Arrighi, E., Grüttemeier, N., Morawietz, N., Sommer, F., and Wolf, P., Multi-Parameter Analysis of Finding Minors and Subgraphs in Edge-Periodic Temporal Graphs, in *SOFSEM 2023: Theory and Practice of Computer Science*, edited by Gąsieniec, L., pages 283–297, Springer, 2023.

[18] Erlebach, T. and Spooner, J. T., A Game of Cops and Robbers on Graphs with Periodic Edge-Connectivity, in *SOFSEM 2020: Theory and Practice of Computer Science: 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20–24, 2020, Proceedings*, page 64–75, Berlin, Heidelberg, 2020, Springer.

[19] Kupferman, O. and Mosheiff, J., Prime languages, volume 240, pages 90–107, 2015, MFCS 2013.

[20] Jecker, I., Kupferman, O., and Mazzocchi, N., Unary Prime Languages, in *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, edited by Esparza, J. and Král', D., volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:12, Dagstuhl, Germany, 2020, Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[21] Jecker, I., Mazzocchi, N., and Wolf, P., Decomposing Permutation Automata, volume abs/2107.04683, 2021.

[22] Bai, C. et al., Predicting the Visual Focus of Attention in Multi-Person Discussion Videos, in *IJCAI 2019*, International Joint Conferences on Artificial Intelligence, 2019.

[23] Kumar, S., Bai, C., Subrahmanian, V., and Leskovec, J., Deception Detection in Group Video Conversations using Dynamic Interaction Networks, in *ICWSM 2021*, International AAAI Conference on Web and Social Media, 2021.

[24] Michail, D., Kinable, J., Naveh, B., and Sichi, J. V., JGraphT - A Java library for graph data structures and algorithms, volume abs/1904.08355, 2019.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelor-/Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vor-gelegt. Sie wurde bisher auch nicht veröffentlicht.

10. Januar 2024