Eötvös Loránd University
Faculty of Science

Márk Hunor Juhász

BSc in Mathematics

Matching problems in temporal graphs

Bachelor thesis

Supervisor: Péter Madarasi

ELTE
EÖTVÖS LORÁND
UNIVERSITY

Budapest, 2025

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Péter Madarasi, who contributed numerous valuable ideas to my thesis and supported me not only during this past semester but throughout the entire academic year. The weekly meetings with him were among the most engaging parts of my studies and played a significant role in fostering my interest in research.

I would also like to thank my friends and my girlfriend for being by my side over the past three years, and for helping me unwind, ensuring that my undergraduate years were filled not only with study, but also with joy and memorable experiences.

Last but not least, I am immensely grateful to my entire family, who have supported and encouraged me for the past 21 years, continually motivating me to pursue my goals.

Table of contents

1 Introduction

Computing a maximum matching in an undirected (static) graph is one of the most fundamental graph-algorithmic problems. It has many different variations and modifications, for example, it can also be extended for temporal graphs, whose topology is subject to discrete changes over time. In these graphs, edges can disappear and then reappear, allowing many real-world optimization problems to be modeled as a maximum matching problem in temporal graphs. However, even the maximum matching problem itself can be defined in various ways within these structures, which are in the focus of the present thesis.

The field of temporal graphs has recently gained significant attention [1, 2, 3] and of course, it is meaningful to study not only matching problems, but also a wide range of other problems in these graphs. For example, one could study parity games on temporal graphs [4], but also many problems defined on static graphs — such as vertex cover or disjoint path problems — can also be investigated in the temporal setting; see [5, 6]. Another recently popular topic is the multistage matching problem [7, 8], which is also closely related to temporal graphs.

In this thesis, we focus on two naturally arising matching problems: Δ- and γ-matching [9, 10]. Additionally, we explore how the seemingly unrelated d-distance matching problem [11, 12] connects to temporal graphs and to the two previous problems. This connection strengthens a hardness result for the γ-matching problem [10]. Namely, we prove that the problem is APX-hard even on bipartite graphs. In Section 3.2, we extend the d-distance matching problem to general graphs, examine how this extended problem relates to the previously discussed ones and we also present a Fixed-Parameter Tracable algorithm parameterized by d.

In Section 4, we examine how the inverse of the d-distance matching problem [12] can be extended to the Δ- and γ-matching problems. Additionally, for the extension to Δ-matchings, we establish several hardness results. In Section 5, we analyze the maximum Δ-matching problem in the case where the underlying static graph of the temporal graph is a tree and prove three hardness results. The Δ-matching problem on trees is meaningful on its own, but as we will see, it is also related to the inverse problems discussed earlier. We prove that this problem becomes NP-hard as soon as every edge can appear at most twice. We also prove that it can be solved in polynomial time if every edge appears at most once. Moreover, we provide an Efficient Polynomial-Time Approximation Scheme for the problem, provided that every edge appears constant times.

2 Δ- and γ-matchings

In this section, we examine the Δ- and the γ-matching problems, discuss some complexity results, and explore the relationship between the two problems. But first, let us present the precise definition of temporal graphs, which originates from the foundational work of Kempe, Kleinberg, and Kumar [13].

Definition 2.1. *A **temporal graph** $\mathcal{G} = (G, \lambda)$ is a pair (G, λ), where $G = (V, E)$ is an underlying (static) graph and $\lambda : E \to 2^{\mathbb{N}} \setminus \{\emptyset\}$ is a **time-labeling function** that specifies which edge is active at what time, that is, an edge $e \in E$ is active at time $\tau \in \mathbb{N}$ if and only if $\tau \in \lambda(e)$. If e is active at time τ, then we call the pair (e, τ) a **time edge** and we say that e **appears** at time τ. The **lifetime** $\mathcal{T}(\mathcal{G}) < \infty$ of a temporal graph is the biggest time-label assigned to an edge, that is, $\mathcal{T}(\mathcal{G}) = \max\{\tau \in \lambda(e) \mid e \in E\}$.*

From this point onward, we will refer to the integers in the set $\{1, \ldots, \mathcal{T}(\mathcal{G})\}$ as time *ticks*. Note that the set of the active edges can be empty in some ticks, that is, there can be a tick τ such that $\tau \notin \lambda(e)$ for any edge e. These graphs are called temporal because each of their "states" is only temporary, lasting for just a single tick.

2.1 The Δ-matching

This problem first appeared in [9] under the name temporal matching. To define matchings in temporal graphs, we need to define what it means for two time edges to be independent. There are several different definitions for this in the literature, and in the case of Δ-matching, it is defined as follows.

Definition 2.2. *Given a non-negative integer $\Delta \in \mathbb{N}$, we call two time edges (e, τ) and (e', τ') **Δ-independent** if the edges e, e' do not share an endpoint or their time-labels τ, τ' are at least Δ time units apart, that is, $|\tau - \tau'| \geq \Delta$. A **Δ-matching** of a temporal graph $\mathcal{G}$ is a set M of pairwise Δ-independent time edges of $\mathcal{G}$.*

The restriction that a vertex cannot be matched more than once within any Δ consecutive ticks, can be thought of as a "recovery" period that an entity (represented by a vertex) must undergo after being paired with another entity. Just as in the case of static graphs, we can also ask how large the maximum matching (in this case, the Δ-matching) is in a temporal graph.

Problem 2.1. *In the **maximum Δ-matching problem**, the input is a temporal graph $\mathcal{G}$ and a non-negative integer $\Delta \in \mathbb{N}$. The goal is to find a Δ-matching in $\mathcal{G}$ with the largest possible cardinality. We refer to the problem of deciding whether a given temporal graph admits a Δ-matching of a given size k as **Δ-matching problem**.*

Remark 2.1. *An alternative way to think about Δ-matchings is that the active time edges in M must form a matching in any consecutive Δ ticks.*

The topic of Δ-matching has been thoroughly explored in [9]. They proved that the maximum Δ-matching problem is APX-complete even if $\Delta = 2, \mathcal{T} = 3$, and every edge of the underlying graph appears only once. They also proved that the Δ-matching problem is NP-complete, even if $\Delta = 2, \mathcal{T} = 5$, and the underlying graph of the input temporal graph is complete. Among several important complexity results, the following theorem will be particularly useful later in this thesis.

Theorem 2.1. *(Mertzios et al. [9]) The maximum Δ-matching problem is NP-hard, even if $\Delta = 2$ and the underlying graph of the input temporal graph is a path.*

In [9], they also presented a $\frac{\Delta}{2\Delta-1}$-approximation algorithm for maximum Δ-matching, and an FPT-algorithm parameterized by the combination of Δ and ν, where ν denotes the size of the maximum matching in the underlying graph G of the input temporal graph. The algorithm runs in $2^{O(\nu\Delta)} \cdot (|V| + \sum_{\tau=1}^{\mathcal{T}} |E_\tau|) \cdot \frac{\mathcal{T}}{\Delta}$ time, where E_τ denotes the set of active edges at time τ. However, this is not the best-known result, as in [14], an algorithm was presented that solves the problem in $\Delta^{O(\nu)} \cdot (|V| + \sum_{\tau=1}^{\mathcal{T}} |E_\tau|)$ time, and hence provided an exponential speedup in terms of Δ.

2.2 The γ-matching

Now, let us consider another definition of independence, which we call γ-independence. The definition is as follows.

Definition 2.3. *Given a temporal graph $\mathcal{G} = (G, \lambda)$, $G = (V, E)$, an edge of the underlying graph $e \in E$ and a non-negative integer $\gamma \in \mathbb{N}$, the **γ-edge of e at time τ** is the set $\{(e, \tau') \mid \tau' \in \{\tau, \ldots, \tau + \gamma - 1\}\}$, where $\{\tau, \ldots, \tau + \gamma - 1\} \subseteq \lambda(e)$. We denote this by $\Gamma_\gamma(e, \tau)$. We say that two γ-edges $\Gamma_\gamma(e, \tau)$ and $\Gamma_\gamma(e', \tau')$ are **γ-independent** if the edges e, e' do not share an endpoint or $\{\tau, \ldots, \tau + \gamma - 1\}$ and $\{\tau', \ldots, \tau' + \gamma - 1\}$ are disjoint. A **γ-matching** of a temporal graph $\mathcal{G}$ is a set M of pairwise independent γ-edges.*

Both Δ- and γ-matching are commonly referred to as *temporal matchings* in the literature, however, we need to make a clear distinction between the two concepts. The problem of maximizing the size of a γ-matching was first introduced in [10] under the same name. The problem is as follows.

Problem 2.2. *In the **maximum γ-matching problem**, the input is a temporal graph $\mathcal{G}$ and a non-negative integer $\gamma \in \mathbb{N}$. The goal is to find a γ-matching in $\mathcal{G}$ with the largest possible cardinality.*

In [10], they worked with a slightly different terminology. They designed a "link stream" generating process by mimicking the behavior of a random moving group of particles. Their main result is showing that the maximum γ-matching problem is NP-hard for every $\gamma \geq 2$. They also provided a kernelization algorithm for the problem parameterized by the size of the solution. On the way to obtaining the kernelization algorithm, they also provided a 2-approximation algorithm for the maximum γ-matching problem. The problem of maximum γ-matching was later studied in [15], where they provided a PTAS for a special case of the problem, and an FPT algorithm for the general case.

However, since the first paper on γ-matching [10] appeared earlier than the first paper on Δ-matching [9], the connection between γ-matching and Δ-matching was only discovered in the latter paper. They showed that there is an easy reduction from the γ-matching to the Δ-matching. This reduction is as follows. For every sequence of γ consecutive time edges starting at tick τ, we introduce only one time edge at tick τ, and set $\Delta = \gamma$. Observe that this is indeed a good reduction, since there is a one-to-one correspondence between the solutions of the two problems. As they underlined in [9], this already implies that deciding whether a given temporal graph admits a Δ-matching of a given size k is NP-complete. However, they were unable to find a reduction in the reverse direction, just as we were not. Nevertheless, some hardness results in [9] can be easily transferred to γ-matchings. For example, they showed that γ-matching is NP-hard and its canonical optimization version is APX-hard even if $\gamma = 2$ and $\mathcal{T} = 4$, which strengthens the hardness result in [10]. By a similar reduction from the d-matching to the γ-matching, we were able to strengthen the hardness results presented in the next section.

3 The d-matching

In this section, we define the d-matching problem, which was first presented in [11] under the name *d-distance matching problem*. We also define the d-b-matching problem, which was first introduced in [12] under the name *d-distance b-matching problem*.

3.1 The d-matching problem and temporal graphs

Originally, the d-matching problem is defined on simple static graphs, but in this section, we discuss it in the concept of temporal graphs and demonstrate that the d-matching problem can even be reduced to the γ-matching problem. But first, consider the definition of the d-matching problem.

Problem 3.1. *In the **maximum d-matching problem**, we are given a bipartite graph $G = (S, T, E)$ with $S = \{s_1, \ldots, s_n\}$, a weight function on the edges $w : E \to \mathbb{R}_+$ and a positive integer $d \in \mathbb{Z}_+$. The goal is to find a maximum-weight subset $M \subseteq E$ of the edges*

such that the degree of every node of S is one in M, and if $s_it, s_jt \in M$, then $|i-j| \geq d$. If a set of edges M satisfies these two conditions, we call it a ***d-matching****.*

In the first paper [11], they showed that the problem is NP-hard, even in the unweighted ($w \equiv 1$) case, but it admits a simple 3-approximation. They also gave an FPT algorithm parameterized by d, showed that the integrality gap of the natural integer programming model is at most $2 - \frac{1}{2d-1}$, and gave an LP-based approximation algorithm for the weighted case with the same guarantee. In [12], the author further studied the problem, and proved that the problem is APX-hard even in the unweighted case. They also gave a $(2-\frac{2}{d})$-approximation algorithm and showed that $2-\frac{2}{d}$ is a tight upper bound on the integrality gap of the natural integer programming model, provided that the nodes in S are considered to be in cyclic order.

At first, it is not clear how this problem is related to temporal graphs, but observe that d-matchings can be formulated as follows.

Remark 3.1. *In the bipartite graph $G = (S, T, E)$, M is a d-matching if we obtain a matching by restricting M to any d consecutive nodes from S. Here, restricting M to a set of nodes X means that we only consider the edges in M, which are induced by X.*

Note that this definition is similar to the one in Remark 2.1. Therefore, with this observation, we got a slightly closer to the concept of temporal graphs. Now, we reduce the maximum d-matching problem to the maximum γ-matching problem and show that the latter problem is APX-hard, even if the underlying graph of the input temporal graph is bipartite. Let the input of the maximum d-matching problem be $G = (S, T, E), S = \{s_1, \ldots, s_n\}, w : E \to \mathbb{R}_+, d \in \mathbb{Z}_+$, and construct the corresponding input $\mathcal{G}' = (G', \lambda'), \gamma' \in \mathbb{N}$ of the maximum γ-matching problem. Let $G' = (V', E)$, where $V' = S \cup T$. We construct the corresponding time-labeling function λ as follows. Let $\lambda(s_it_j) = \{i, \ldots, i+d-1\}$, and let $\gamma = d$. Observe that with this reduction, there is a one-to-one correspondence between the solutions of the two problems.

Corollary 3.1. *The maximum γ-matching problem is APX-hard, even if the underlying graph of the input temporal graph is bipartite.*

Corollary 3.2. *The maximum d-matching problem can be reduced to the maximum Δ-matching problem.*

In [12], the author also studied a different problem related to d-matching, the d-distance b-matching problem, which we refer to as d-b-matching.

Problem 3.2. *In the* ***maximum d-b-matching problem****, we are given a bipartite graph $G = (S, T, E)$ with $S = \{s_1, \ldots, s_n\}$, a weight function on the edges $w : E \to \mathbb{R}_+$, a positive integer $d \in \mathbb{Z}_+$ and a degree bound function $b : S \cup T \to \mathbb{Z}_+$. The goal is to find a*

maximum-weight subset $M \subseteq E$ of the edges satisfying the following two conditions. First, the degree of every node $v \in S \cup T$ is at most $b(v)$ in M. Second, if $s_i t, s_j t \in M$, then $|i - j| \geq d$. If a set of edges M satisfies these two conditions, we call it a ***d-b-matching****.*

It is easy to see that we get back the maximum d-matching problem when $b(s) = 1$ for every $s \in S$ and $b(t) = \infty$ for every $t \in T$. Hence, this problem is NP-hard as well. In the cyclic version of the problem, the nodes in S are considered to be in cyclic order. In [12], they prove that $2 - \frac{1}{d}$ is a tight upper bound on the integrality gap of the natural integer programming model for the cyclic maximum d-b-matching problem provided that $2d - 1$ divides the size of S.

Also in this paper, a different aspect of the problem is considered, in which the goal is to find a permutation of S maximizing the weight of the optimal d-b-matching. They prove that such a permutation can be found in polynomial time — even though the maximum d-matching problem itself is NP-hard and also hard to approximate arbitrarily. For d-b-matchings, however, they prove that finding the best permutation is NP-hard, even when $b(s) = 2$ for all $s \in S$ or $d = 2$, and they give e-approximation algorithms for both the cyclic and the non-cyclic cases, where e is the Euler's number. We refer to the problem of finding the best permutation as the *inverse* of the original problem. In Section 4, we further discuss the inverse of the maximum d-b-matching problem and examine what can be stated about the inverse of the Δ-matching and γ-matching problems. But first, let us present a variant of the maximum d-matching problem.

3.2 An adaptation of d-matchings to general graphs

Note that the alternative definition in Remark 3.1 still makes sense even if the graph is non-bipartite; however, in this case, we must define an ordering for all the vertices. Thus, we can introduce a variant of the d-matching problem defined for general graphs, which resembles the bipartite case.

Definition 3.1. *In the* ***non-bipartite d-matching problem****, we are given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, a weight function on the edges $w : E \to \mathbb{R}_+$ and a positive integer $d \in \mathbb{Z}_+$. The goal is to find a maximum-weight subset $M \subseteq E$ of the edges such that we obtain a matching by restricting M to any d consecutive points from V. If a set of edges M satisfies this condition, we call it a* ***non-bipartite d-matching****.*

However, despite this being a variant of the problem defined on bipartite graphs, we did not find a reduction either from the original problem to the non-bipartite one or in the reverse direction. Nevertheless, we can prove that the non-bipartite d-matching problem is NP-hard by reducing another NP-hard problem to it. The other problem is the *double matching problem*, which was used in [11] to prove that the d-matching problem is NP-hard. The definition of the double matching problem is as follows.

Problem 3.3. *In the **double matching problem**, we are given a bipartite graph $G = (S, T, E)$ and two sets $S_1, S_2 \subseteq S$ such that $S_1 \cup S_2 = S$. The goal is to find a maximum-size subset $M \subseteq E$ for which both $M \cap E_1$ and $M \cap E_2$ are matchings, where E_i denotes the edges induced by S_i and T for $i = 1, 2$.*

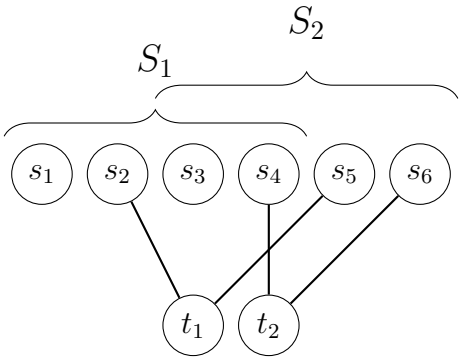
Theorem 3.1. *The non-bipartite d-matching problem is NP-hard, even in the unweighted case.*

Proof. Madarasi [11] proved that the double matching problem is NP-hard even if $|S_1| = |S_2|$. Now, we reduce this problem to the non-bipartite d-matching problem in the following way. Let the input of the double matching problem be $G = (S, T, E)$ and $S_1, S_2 \subseteq S$, such that $S_1 \cup S_2 = S$ and $|S_1| = |S_2|$. We can now construct the corresponding input of the non-bipartite d-matching problem. Let the new input graph be $G' = (V', E)$, the weight function on the edges $w' \equiv 1$ and the positive constant $d' = |S_1| + |T| + |D_1|$, where D_1 is the set of $|S_1 \setminus S_2|$ dummy nodes. The set of nodes in the new input is $V' = (S_1 \setminus S_2) \cup D_1 \cup (S_1 \cap S_2) \cup T \cup D_2 \cup (S_2 \setminus S_1)$, where D_2 is a copy of D_1. In the previous expression, the vertices corresponding to each set follow the same order as the sets themselves in the previous expression. Within each set, the order of the vertices is as follows. In $(S_1 \setminus S_2), (S_1 \cap S_2)$ and $(S_2 \setminus S_1)$ the order of the nodes is the same as before, in T, D_1 and D_2 the order of the nodes is arbitrary. The edge set of G' is the same as that of G. The original input G of the double matching problem is illustrated in Figure 1a and the constructed G' is illustrated in Figure 1b.

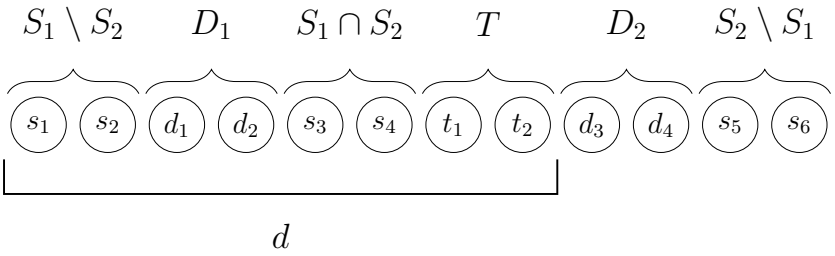
Let the solution of the constructed non-bipartite d-matching problem be M^*. Observe that M^* is a maximum-size subset $M^* \subseteq E$ for which both $M^* \cap E_1$ and $M^* \cap E_2$ are matchings. Also observe that this is the only constraint in the constructed non-bipartite d-matching problem and hence there is a one-to-one correspondence between the solutions of this problem and the solutions of the double matching problem. □

Since the problem is NP-hard, we can only hope to approximate the optimal solution. But before moving on to the approximation algorithm, let us observe a few properties of the feasible solutions. First of all, it can be assumed that every edge e with endpoints v_i, v_j, where $|i - j| \geq d$, are included in the solution. In these cases we say that the edge e *jumps* a distance at least d. Second, observe that without edges that jump a distance at least d, a feasible solution M to the problem is the union of node-disjoint paths. Indeed, if we only consider the nodes and the edges of the solution $(G|_M)$, every node has a degree of at most two, since each node can only have one right and one left neighbor. With these observations, we are now ready to present a 2-approximation algorithm to the problem.

The algorithm is very simple: it first includes all the edges into the solution M, which jump a distance at least d, then it finds a maximum-weight matching in G, and includes its edges into M. As we observed, the optimal solution M^* in G is a set of node-disjoint paths



(a) An instance G of the double matching problem.



(b) The input G' of the non-bipartite d-matching problem constructed from G.

Figure 1: Illustration of the construction of G' in the proof of Theorem 3.1.

— apart from the edges that jump a distance at least d. Therefore, $M^* = (M_1^* \cup M_2^*) \cup J$, where M_1^* and M_2^* are edge-disjoint matchings and J is the set of all edges that jump a distance at least d, where $J \cap (M_1^* \cup M_2^*) = \emptyset$. Observe that $w(J) + \max(w(M_1^*), w(M_2^*)) \geq \frac{1}{2} w(M^*)$, where $w(X)$ denotes the sum of the edge-weights in X. By definition, the set M produced by the algorithm satisfies that $w(M) \geq w(J) + \max(w(M_1^*), w(M_2^*)) \geq \frac{1}{2} w(M^*)$, thus the algorithm is indeed 2-approximating.

As previously mentioned, Madarasi [11] provided an FPT algorithm for the (weighted) d-matching problem, parameterized by d. In what follows, this approach is adapted to present an FPT algorithm for the non-bipartite d-matching problem, also parameterized by d.

First, if there are parallel edges in the graph G, we reduce the graph by keeping only one edge from each set of parallel edges — since at most one of them can be selected into the solution. This reduction can be done in $O(m)$ steps, where m is the number of edges. Second, if there are some edges in G which jump a distance d, we include them in the solution and reduce the graph by excluding these edges. This reduction also runs in $O(m)$ steps. After these reductions, the degree of each node in G is at most $2d - 2$.

In what follows, a dynamic programming approach is presented to solve the problem (after the previous reductions) in $O(nd(2d-1)^d)$ steps. Let the nodes of G be $V = \{v_1, \ldots, v_n\}$, in this order. For $i \geq d$, let $f(v_i, z_1, \ldots, z_d)$ denote the weight of the

maximum-weight d-matching if the problem is restricted to the first i nodes and v_{i-j+1} is *assigned* to z_j for $j \in [d]$, where $z_j \in (N(v_{i-j+1}) \cap \{v_{i-j}, v_{i-j-1}, \dots, v_{i-j-d+2}\}) \cup \{\varnothing\}$. Here, $N(v)$ denotes the neighbors of v in G, and $\varnothing$ denotes that we assigned nothing to a node. Assigning v_{i-j+1} to z_j, where both v_{i-j+1} and z_j are nodes of G, means that we include the edge $v_{i-j+1}z_j$ into the solution. If $v_{i-j+1} \in V$ and $z_j = \varnothing$, then assigning v_{i-j+1} to z_j means that for every $w \in (N(v_{i-j+1}) \cap \{v_{i-j}, v_{i-j-1}, \dots, v_{i-j-d+2}\})$, the edge wv_{i-j+1} is not included into the solution. Let us denote the set of the "possible left neighbors" of v_{i-j+1} by $L(v_{i-j+1}) = (N(v_{i-j+1}) \cap \{v_{i-j}, v_{i-j-1}, \dots, v_{i-j-d+2}\}) \cup \{\varnothing\}$.

By definition,

$$f(v_i, z_1, \dots, z_d) = -\infty \tag{1}$$

if a node appears multiple times among $z_1, \dots, z_d$, since it yields an invalid solution. However, note that $\varnothing$ can occur multiple times among the neighbors. Equation (1) also holds if there exists a node v_{i-j+1} such that its left neighbor is $z_j = v_k$ and v_k also has a left neighbor $z_{i-k+1} = v_\ell$, where $|(i-j+1)-\ell| < d$. In this case, v_k has two neighbors, which are closer to each other than d, hence it also yields an invalid solution. Thus, checking whether $f = -\infty$ can be done in $O(d)$ steps for each combination of $v_i, z_1, \dots, z_d$.

If $f(v_i, z_1, \dots, z_d) \neq -\infty$, it can be calculated by the following recursive formula.

$$f(v_i, z_1, \dots, z_d) = \begin{cases} \displaystyle\sum_{j=1}^{d} w(v_{d-j+1}z_j) & \text{if } i = d, \\ \displaystyle w(v_iz_1) + \max_{\widehat{z} \in L(v_{i-d})} f(v_{i-1}, z_2 \dots, z_d, \widehat{z}) & \text{if } i > d, \end{cases} \tag{2}$$

where $i \geq d$, $v_i \in V$, $z_j \in L(v_{i-j+1})$ for $j \in [d]$ and $f(v_i, z_1, \dots, z_j) \neq -\infty$. To see that this recursion holds, observe that in the first case, by definition, $f(v_i, z_1, \dots, z_d)$ is the weight of the non-bipartite d-matching $\{v_jz_{d-j+1} : j \in [d]\}$. In the second case, v_i must be mapped to z_1 and we want to find the maximum-weight non-bipartite d-matching on the first $i-1$ nodes which maps v_{i-j+1} to its left neighbor z_j for $j \in \{2, \dots, d\}$. To this end, we want to find a left neighbor $\widehat{z} \in L(v_{i-d})$, to be assigned to node v_{i-d}, which maximizes $f(v_{i-1}, z_2, \dots, z_d, \widehat{z})$. Observe that the solution to the problem, by definition, is

$$\max\{f(v_n, z_1, \dots, z_d) : z_j \in L(v_{n-j+1}) \text{ for } j \in [d]\}, \tag{3}$$

which we can obtain by solving all the subproblems.

The number of subproblems is $O(n(2d-1)^d)$, since every node can have at most $2d-1$ neighbors (including $\varnothing$). Equation (2) gives a way to compute $f(v_i, z_1, \dots, z_d)$ in $O(d)$ steps if the subproblems are computed in appropriate order. That is, all the necessary $f(v_{i-1}, z'_1, \dots, z'_d)$ values are already calculated. Therefore, the number of steps to compute all the subproblems is $O(nd(2d-1)^d)$ and the solution can be computed in $O((2d-1)^d)$ steps by (3). This means that the algorithm is indeed fixed-parameter tractable with respect to the parameter d.

4 Inverse problems of the maximum Δ-matching

As it is discussed in [12], it is a natural question whether we can find a permutation of S maximizing the weight of the optimal d-b-matching. Formally, let M^*_π denote an optimal d-b-matching under the permutation π of S. We want to find a permutation of S and a d-b-matching M^* with respect to this permutation such that $w(M^*) = \max_{\pi\in\mathcal{C}} w(M^*_\pi)$, where $\mathcal{C}$ is the set of all permutations of S and w is the weight function on the edges. We refer to this as the *inverse* of the original problem. In [12], they present a polynomial-time algorithm that finds an optimal permutation and an optimal d-b-matching under this permutation, provided that $b(s) = 1$ for all $s \in S$ and $b(v) = \infty$ for all $v \in T$. They also prove that the analogous problem is NP-hard even if $b \equiv 2$ or $d = 2$. Following these results, one may ask what can be said about the inverse of the Δ-matching and the γ-matching.

In Section 4.1, we examine a few unweighted inverse definitions of the maximum Δ-matching problem and briefly discuss why there is not much to say about the inverse of the maximum γ-matching problem. In Section 4.2 we explore a few weighted cases, and then in Section 4.3, we examine a new edge coloring problem related to this topic, the bagpiper edge-coloring.

4.1 Unweighted inverse problems

What follows is a definition for the inverse of the maximum Δ-matching problem.

Problem 4.1. *We are given a graph $G = (V, E)$ and a positive integer $k \in \mathbb{Z}_+$. The goal is to assign an integer $\lambda(e) \in [k]$ to each edge of G in such a way that the solution of the maximum Δ-matching problem on the temporal graph $\mathcal{G} = (G, \lambda)$ is maximized.*

Formally, let M^*_λ denote a maximum Δ-matching under the assignment function (which is also the time-label function) λ. We want to find a labeling function $\lambda : E \to [k]$ and a Δ-matching M^* with respect to this labeling function such that $|M^*| = \max_{\lambda\in\mathcal{F}} |M^*_\lambda|$, where $\mathcal{F}$ is the set of all functions $E \to [k]$. For the sake of simplicity, at first, let $\Delta = 2$, and we deal with the general case later.

Lemma 4.1. *There is always an optimal solution λ^* to Problem 4.1, where $\lambda^*(e)$ is odd for all $e \in E$, provided that $\Delta = 2$.*

Proof. Let λ be an optimal solution, and we now construct λ^*, where λ^* is also optimal, but only uses odd time-labels. For all the edges $e \in E$, where $\lambda(e) = 2i - 1$ for some $i \in \{1, \dots, \lceil k/2 \rceil\}$, let $\lambda^*(e) = \lambda(e)$. If there is an edge e for which $\lambda(e) = 2$, then let $\lambda^*(e) = 1$. Since $\Delta = 2$, this change can only increase the size of the optimal solution of the maximum Δ-matching problem. After we changed all the 2-s to 1-s, we can move on to the next even number. Now, if we have an edge $e \in E$, for which $\lambda(e) = 4$, then let

$\lambda^*(e) = 3$. Since we do not have any time-labels with the value of 2, this change can also only increase the size of the optimal solution. Continue this for all the even numbers (in increasing order), that is, if $\lambda(e) = 2i$ for some $i \in \{1, \dots, \lfloor k/2 \rfloor\}$, then let $\lambda^*(e) = 2i - 1$. Since every change can only affect the size of the optimal solution in a positive way, λ^* is at least as good as λ. □

After this lemma, the problem can also be formulated as follows: find a maximum $\lceil k/2 \rceil$-edge-colorable subset of edges in G. Indeed, if we only use every second integer as time-labels, the $\Delta = 2$ constraint is exactly the constraint in the standard edge-coloring problem, that is, the edges of the same color (same time-label) cannot be adjacent. In the general case, where Δ is not necessarily 2, we can make a very similar observation.

Lemma 4.2. *There is always an optimal solution λ^* to Problem 4.1, where $\lambda^*(e)$ can be written as $\lambda^*(e) = j\Delta + 1$ for some $j \in \{0, \dots, \lfloor k/\Delta \rfloor\}$ for each edge $e \in E$.*

Proof. The proof is the same as in Lemma 4.1, the only difference is that we use different values in λ^*. □

After this observation, the general problem can also be formulated as follows: we are looking for a maximum $\lceil k/\Delta \rceil$-edge-colorable subset of edges in G. This problem is NP-hard, even if $\Delta = 2$, as it is proven in [16, 17, 18]. However, if the graph G is bipartite, we can solve the problem in polynomial time. Note that in [18], they have already solved this problem, but in this thesis we present a different solution.

Theorem 4.1. *Problem 4.1 can be solved in polynomial time if the graph G is bipartite.*

Proof. Observe that due to Kőnig's edge-coloring theorem, the problem can be formulated as follows. We aim to find a set of edges $M \subseteq E$, such that the maximum degree in $G|_M$ is at most $\lceil k/\Delta \rceil$. We can write the corresponding integer programming problem in the form of $Ax \leq b$, $\max \mathbb{1}x$, where b is a vector, in which all coordinates are $\lceil k/\Delta \rceil$, A is the incidence matrix of G, where the columns correspond to the edges, and $x \in \{0, 1\}$ indicates if a column (edge) is chosen into the solution or not. Since G is bipartite, the matrix of this IP-problem is totally unimodular. The right-hand side is a vector containing integers, therefore the theorem holds [19, 20]. □

Let us slightly alter the previous definition to get to the second naturally arising inverse problem.

Problem 4.2. *We are given a graph $G = (V, E)$ and a positive integer $k \in \mathbb{Z}_+$. The goal is to assign some integers $\lambda(e) \subseteq \{1, \dots, k\}$ to each edge of G in such a way that the solution of the maximum Δ-matching problem on the temporal graph $\mathcal{G} = (G, \lambda)$ is maximized.*

Formally, let M^*_λ denote a maximum Δ-matching under the assignment function (which is also the time-label function) λ. We want to find a labeling function $\lambda : E \to 2^{[k]}$ and a Δ-matching M^* with respect to this labeling function such that $|M^*| = \max_{\lambda \in \mathcal{F}} |M^*_\lambda|$, where $\mathcal{F}$ is the set of all functions $E \to 2^{[k]}$.

If there is no upper bound on the time-labels that λ can assign to an edge, the problem is trivial, since $\lambda(e) = \{1, \ldots, k\}$ is always optimal. In fact, the size of the optimal solution of the maximum Δ-matching is always $|M| \cdot \lceil k/\Delta \rceil$, where M is a maximum matching in G. Let us introduce a *capacity constraint* $c \in \mathbb{Z}_+$ on the time-labels that λ can assign to an edge. That is, $|\lambda(e)| \leq c$ for all $e \in E$. Note that with $c = 1$, we obtain Problem 4.1.

Remark 4.1. *Problem 4.2 with this capacity constraint can be solved as follows. Instead of assigning a capacity of c to each edge, include each edge in the graph c times as parallel edges, and solve Problem 4.1 on this graph.*

With this observation, it is easy to see that even with capacity constraint, there is always an optimal λ^* function, where $\lambda^*(e) = j\Delta + 1$ for some $j \in \{0, \ldots, \lfloor k/\Delta \rfloor\}$ for each edge $e \in E$. We can also observe that in some special cases, the problem is easily solvable.

Theorem 4.2. *Problem 4.2 with capacity constraints can be solved in polynomial time, provided that $c \geq \lfloor k/\Delta \rfloor$ and $\Delta > 0$.*

Proof. We solve this problem based on Remark 4.1. Let λ^* be an optimal assignment function, where for every edge e, $\lambda^*(e) = j\Delta + 1$ for some $j \in \{0, \ldots, \lfloor k/\Delta \rfloor\}$. Let M_{λ^*} be the corresponding optimal solution to the maximum Δ-matching problem under the labeling function λ^*. For every $j \in \{0, \ldots, \lfloor k/\Delta \rfloor\}$, let $M^{j\Delta+1}_{\lambda^*}$ denote the set of edges, where $\lambda^*(e) = j\Delta + 1$, and $(e, j\Delta + 1) \in M_{\lambda^*}$. Observe that for every j, $M^{j\Delta+1}_{\lambda^*}$ is a matching, since $\Delta > 0$. Let P denote the maximum matching in the graph. Obviously, $|P| \geq |M^{j\Delta+1}_{\lambda^*}|$ for every $j \in \{0, \ldots, \lfloor k/\Delta \rfloor\}$. Now construct an assignment function λ, where $\lambda(e^j) = j\Delta + 1$ for every $e \in P$ and every $j \in \{0, \ldots, \lfloor k/\Delta \rfloor\}$, where $\{e^0, \ldots, e^c\}$ is an arbitrary ordering of the edges parallel to e (including e). For every other edge $f \notin P$, let $\lambda(f) = 1$ and let M_λ denote a corresponding optimal solution of the maximum Δ-matching under the labeling function λ. Observe that $|M^{j\Delta+1}_\lambda| = |P| \geq |M^{j\Delta+1}_{\lambda^*}|$ for every $j \in \{0, \ldots, \lfloor k/\Delta \rfloor\}$. Since $|M_{\lambda^*}| = \sum_{j=0}^{\lfloor k/\Delta \rfloor} |M^{j\Delta+1}_{\lambda^*}| \leq \sum_{j=0}^{\lfloor k/\Delta \rfloor} |P| = |M_\lambda|$, we can be sure that λ is at least as good as λ^*. Observe that in order to construct λ, we only need to be able to find the maximum matching P in the graph, which we can do in polynomial time, thus the theorem holds. □

The question may arise as to why we are not addressing the inverse of the maximum γ-matching problem as well. The reason for this is that if we attempt to rewrite Problem 4.1 in terms of γ-matching, we arrive at the following problem.

Problem 4.3. *We are given a graph $G = (V, E)$ and a positive integer $k \in \mathbb{Z}_+$. The goal is to assign some integers $\lambda(e) \subseteq [k]$ to each edge of G in such a way that the solution of the maximum γ-matching problem on the temporal graph $\mathcal{G} = (G, \lambda)$ is maximized.*

It is easy to see that, due to the definition of γ-matching, there is always an optimal labeling function λ^* that assigns numbers only to the edges of the largest matching in G.

4.2 Weighted inverse problems

Let us return to Problem 4.1. As we have already established, the problem is solvable for bipartite graphs, while in the general case, it is NP-hard. However, we can observe that for bipartite graphs, the problem was solved almost "too easily", giving the impression that even more challenging problems could be tackled. Therefore, we now consider the weighted versions of the problem, specifically three different weighted cases. In the first case, we only have weights on the edges, but since we are minimizing, we call them costs.

Problem 4.4. *We are given a graph $G = (V, E)$, a positive integer $k \in \mathbb{Z}_+$, and a cost function on the edges $c : E \to \mathbb{R}_+$. The goal is to assign an integer $\lambda(e) \in [k]$ to each edge of G in such a way that the solution of the maximum Δ-matching problem on the temporal graph $\mathcal{G} = (G, \lambda)$ is maximized, while the cost of the edges in the solution is minimized.*

Formally, let M^*_λ denote a maximum Δ-matching under the assignment function (which is also the time-label function) λ. We want to find a labeling function $\lambda : E \to [k]$ and a Δ-matching M^* with respect to this labeling function such that $|M^*| = \max_{\lambda \in \mathcal{F}} |M^*_\lambda|$, where $\mathcal{F}$ is the set of all functions $E \to [k]$. We also have to satisfy the following condition: $\text{cost}(M^*) = \min_{M_s \in S} \text{cost}(M_s)$, where S is the set of all optimal sized solutions to the maximum Δ-matching problem. Here, $\text{cost}(M) = \sum_{e \in M} c(e)$. It turns out, that we can solve this problem in polynomial time.

Theorem 4.3. *Problem 4.4 is solvable in polynomial time, provided that the graph G is bipartite.*

Proof. We solve the problem by formulating two integer programming (IP) problems that precisely describe it. Then, we show that the matrices of both IP problems are totally unimodular (TU), and since the right-hand sides consist of integer vectors, we can find a solution in polynomial time [19, 20].

The first IP problem is the same as the one in the proof of Theorem 4.1. We already proved that we can solve this problem in polynomial time, hence we can calculate the size of an optimal solution in polynomial time. We can use this information in the second IP problem, where we look for a solution with this size, but with the least cost.

The second problem can be formulated as follows: $Bx \leq b$, $\max cx$, where $B = \begin{bmatrix} A \\ \mathbb{1} \end{bmatrix}$, $b = \begin{bmatrix} b_A \\ b_{\mathbb{1}} \end{bmatrix}$, and A is the incidence matrix of G, where the columns correspond to the edges. Every coordinate in b_A is exactly $\lceil k/\Delta \rceil$, and $b_{\mathbb{1}}$ has only one coordinate, which is the previously calculated size of the optimal solution (an integer). Moreover, the vector c contains the costs of the edges, and the edges appear in the same order as in the matrix A. It is easy to see that the right-hand side is an integer vector. Since A is the incidence matrix of a bipartite graph, B is a well-known network matrix, hence it is totally unimodular. □

Thus, with this edge-cost definition, the problem was once again solvable. Now, let us move on to the other edge-cost definitions, which are as follows.

Problem 4.5. *We are given a graph $G = (V, E)$, a positive integer $k \in \mathbb{Z}_+$ and a cost function on the labels $c : [k] \to \mathbb{R}_+$. The goal is to assign an integer $\lambda(e) \in [k]$ to each edge of G in such a way that the solution of the maximum Δ-matching problem on the temporal graph $\mathcal{G} = (G, \lambda)$ is maximized, while the cost of the labels in the solution is minimized.*

In the first problem, we gave costs to the edges and now we gave costs to the labels. In the third problem we assign a cost to every edge-label pair.

Problem 4.6. *We are given a graph $G = (V, E)$, a positive integer $k \in \mathbb{Z}_+$, and a cost function on the edge-label pairs $c : E \times [k] \to \mathbb{R}_+$. The goal is to assign an integer $\lambda(e) \in [k]$ to each edge of G in such a way that the solution of the maximum Δ-matching problem on the temporal graph $\mathcal{G} = (G, \lambda)$ is maximized, while the cost of the edge-label pairs in the solution is minimized.*

Now, we will present complexity results for a closely related problem, and using the methods applied in the proofs, similar results could be obtained for the previous two problems as well. However, we omit these from the thesis.

4.3 The bagpiper edge-coloring problem

It is somewhat confusing that in the previous problems we only focus on optimal-sized solutions. However, this is necessary, since we aim to optimize for the maximum Δ-matching problem, while the cost is only a secondary consideration. If we did not require the solution to be of optimal size, we would need to penalize the solution size in some way. To avoid this, we can also attempt the following definition.

Problem 4.7. *In the **bagpiper edge-coloring problem**, we are given a graph $G = (V, E)$ and two positive integers $\Delta, k \in \mathbb{Z}_+$. The goal is to assign an integer $\lambda(e) \in [k]$ to*

some edges $e \in E$, such that no two adjacent edges have been assigned numbers that are closer than Δ. We call a labeling function λ a ***bagpiper edge-coloring*** *of the graph if it satisfies the condition above.*

For the sake of simplicity, let us assume that λ assigns something to each edge, and if we do not want to assign an integer to some edge e, we say that $\lambda(e) = \varnothing$. In this definition, k denotes the number of colors, that is, each integer $i \in [k]$ corresponds to a color. Note that the standard edge-coloring problem is the special case of the bagpiper edge-coloring problem, where $\Delta = 1$ and $\lambda(e) \neq \varnothing$ for each edge e.

To see the connection between this problem and the previous weighted inverse definitions, let us consider the optimization version of this problem. As before, let us start with the edge-weighted case. It is important to maximize here; otherwise, we would need to penalize omitted edges or deal with negative weights.

Problem 4.8. *In the first weighted case of the bagpiper edge-coloring problem, we are given a graph $G = (V, E)$, two positive integers $\Delta, k \in \mathbb{Z}_+$ and a weight function $w : E \to \mathbb{R}$ on the edges. The goal is to find a bagpiper edge-coloring λ of the graph G, where the sum of the weights of the colored edges is as big as possible.*

Let Λ denote the set of bagpiper edge-colorings λ of the graph G. We want to find the coloring $\lambda^* \in \Lambda$, such that $w(\lambda^*) = \max_{\lambda \in \Lambda} w(\lambda)$, where $w(\lambda) = \sum_{e \in E : \lambda(e) \neq \varnothing} w(e)$. Note that in the special case $w \equiv 1$ the problem is equivalent to Problem 4.1. Also observe that Lemma 4.2 remains true in this weighted case of the bagpiper edge-coloring, for the same reason as before.

Theorem 4.4. *Problem 4.8 can be solved in polynomial time, provided that the input graph G is bipartite.*

Proof. Since Lemma 4.2 holds, due to Kőnig's edge-coloring theorem, we can use a similar proof to the proof of Theorem 4.1. We can write the corresponding IP problem in the form of $Ax \leq b$, $\max wx$, where b is a vector, in which all coordinates are $\lceil k/\Delta \rceil$, A is the incidence matrix of G, and $x \in \{0, 1\}$ indicates if an edge is selected or not. The vector w contains the weights of the edges, the i-th coordinate of the vector is the weight of the edge corresponding to the i-th column of A. Since G is bipartite, the matrix of the IP problem is totally unimodular, and since the vector b contains only integers, the theorem holds [19, 20]. □

Since we were able to solve this problem, the question arises whether the reformulation of the other two problems introduced in the previous section (Problems 4.5, and 4.6) can also be solved. First, we examine the exact reformulation of these problems, and then we prove that one of them is NP-hard on bipartite graphs.

Problem 4.9. *In the second weighted case of the bagpiper edge-coloring problem, we are given a graph $G = (V, E)$, two positive integers $\Delta, k \in \mathbb{Z}_+$ and a weight function $w : [k] \to \mathbb{R}$ on the labels. The goal is to find a bagpiper edge-coloring λ of the graph G, where the sum of the weights of the labels assigned to the edges (with multiplicity) is as big as possible.*

Once again, let Λ denote the set of bagpiper edge-colorings λ of the graph G. We want to find a coloring $\lambda^* \in \Lambda$, such that $w(\lambda^*) = \max_{\lambda \in \Lambda} w(\lambda)$, where $w(\lambda) = \sum_{e \in E : \lambda(e) \neq \varnothing} w(\lambda(e))$. Now, let us present the third weighted case of the bagpiper edge-coloring problem, which turns out to be NP-hard.

Problem 4.10. *In the third weighted case of the bagpiper edge-coloring problem, we are given a graph $G = (V, E)$, two positive integers $\Delta, k \in \mathbb{Z}_+$ and a weight function $w : E \times [k] \to \mathbb{R}$ on the edge-label pairs. The goal is to find a bagpiper edge-coloring λ of the graph G, where the sum of the weights of the edge-label pairs in the coloring is as big as possible.*

Theorem 4.5. *Problem 4.10 is NP-hard, even if the input graph is bipartite and $\Delta = 2$.*

Proof. We reduce the maximum Δ-matching problem on paths to Problem 4.10, since in [9], they proved that the former problem is NP-hard, even if $\Delta = 2$ (see Lemma 2.1). Let the input temporal graph of the maximum Δ-matching problem be $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$ is a path. We construct the bipartite graph $G' = (V', E')$ and the weighting function w' in such a way that if we could solve Problem 4.10 on G' with respect to w' in polynomial time, then we could solve the above mentioned NP-hard Δ-matching problem in polynomial time. Note that we still need two more input parameters for Problem 4.10: Δ' and k'. For this purpose, we use $\Delta' = \Delta$ and $k' = \mathcal{T}$, where $\mathcal{T}$ is the lifetime of $\mathcal{G}$.

We construct G' by modifying G in the following way. For every node $v \in V$, let $v' \in V'$. For every edge $uv \in E$, add $|\lambda(uv)|$ parallel edges between the nodes u' and v'. Since G was a path, G' is a path with parallel edges, hence it is bipartite. All that remains is to construct the appropriate weight function w'. Let $e^1_{u'v'}, \dots, e^{|\lambda(uv)|}_{u'v'}$ be an arbitrary ordering of the parallel edges between u' and v' in G', and let $\lambda(uv)^1, \dots, \lambda(uv)^{|\lambda(uv)|}$ be an arbitrary ordering of the time-labels assigned to the edge $uv \in E$ in $\mathcal{G}$. For every $u'v' \in E'$, every $i \in [|\lambda(uv)|]$, and every $j \in [k]$, let

$$w'(e^i_{u'v'}, j) = \begin{cases} 1 & \text{if } i = j, \\ -\infty & \text{if } i \neq j. \end{cases} \tag{4}$$

Let λ^* be an optimal solution to Problem 4.10 in G' with respect to Δ', k' and w'. Observe that either $\lambda^*(e^i_{u'v'}) = \varnothing$ or $\lambda^*(e^i_{u'v'}) = i$. Also observe that we constructed G' and w' in such a way that there is a one-to-one correspondence between the solutions of the two problems. Moreover, from an optimal solution to Problem 4.10 in G', we

can easily construct an optimal solution M^* to the maximum Δ-matching problem in $\mathcal{G}$ in polynomial time as follows. Let $M^* = \{(uv, i) : \lambda^*(e^i_{u'v'}) = i : u'v' \in E', i \in [|\lambda(uv)|]\}$. Due to the one-to-one correspondence, this is a feasible and optimal solution to the problem. Therefore the theorem holds. □

We saw that Problem 4.10 is NP-hard on bipartite graphs. The question may arise whether the problem remains NP-hard if the input graph is a tree (or a path).

Remark 4.2. *From the proof of Theorem 4.5 it is clear that if the maximum Δ-matching was NP-hard on trees (or paths), even if every edge appeared at most once, then Problem 4.10 would be NP-hard, even if the input graph G was a tree (or a path).*

This question is quite interesting since many NP-hard problems become solvable on trees, so it would be quite surprising if this problem remained hard (even on paths). This is precisely why we began to investigate this question as well. During the course of our research, interesting complexity results also emerged in connection with the maximum Δ-matching problem on trees, which will be discussed in the following section.

5 Δ-matching on trees

The Δ-matching problem has been widely studied, but there are only a few results concerning the case where the underlying graph is a tree. Based on Remark 4.2, if we could prove that the maximum Δ-matching problem is NP-hard on trees, even if every edge appears at most once, we could conclude that Problem 4.10 is NP-hard even on trees. In Section 5.1, we examine the case where every edge can appear at most twice, and prove that this problem is NP-hard. However, in Section 5.2, we prove that in the case of single edge appearances, the problem can be solved in polynomial time, thus we still do not know the hardness of Problem 4.10. Nevertheless, in Section 5.3, we prove that the maximum Δ-matching problem admits an Efficient Polynomial-Time Approximation Scheme, provided that the underlying graph G of the input temporal graph $\mathcal{G}$ is a tree and Δ is constant.

5.1 Multiple edge appearances

A natural question is whether we can solve the maximum Δ-matching problem on trees when every edge can appear multiple times. From [9], we know that if an edge can appear arbitrarily many times, the problem becomes NP-hard. Therefore, the question is where the threshold lies, i.e., what is the smallest number of times an edge can appear while the problem still remains solvable. In this section, we prove that even with just 2 appearances, the problem becomes NP-hard.

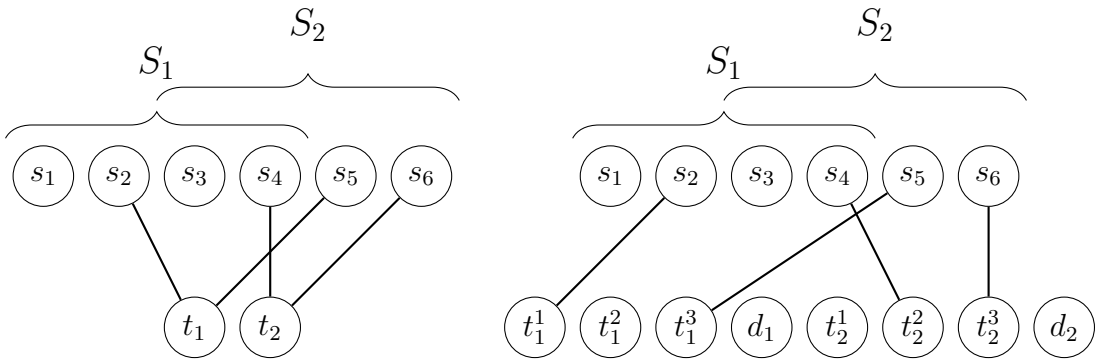


Figure 2: Example of the construction of G' in the proof of Theorem 5.1. The graph on the left is the input G of the double matching problem, while the one on the right is the constructed bipartite graph G'.

Theorem 5.1. *Maximum Δ-matching is NP-hard even if $\Delta = 2$, the underlying graph of the temporal graph is a tree and every edge appears at most twice.*

To prove Theorem 5.1, we need a slightly stronger version of Theorem 3 from [12], which we obtain by observing that the constructed bipartite graph in the proof of the latter theorem has the property that the degree of every node $s \in S$ is exactly 2.

Lemma 5.1. *(Madarasi [11]) The unweighted double matching problem is NP-hard to α-approximate for any $\alpha < \frac{950}{949}$, even if the degree of every node $s \in S$ is exactly 2. Hence, the unweighted double matching problem is NP-hard, even if the degree of every node $s \in S$ is exactly 2.*

Now, we will prove Theorem 5.1 by reducing the NP-hard problem we just obtained to the maximum Δ-matching problem on trees, where $\Delta = 2$ and every edge appears at most twice.

Proof of Theorem 5.1. In the unweighted double matching problem, we are given a bipartite graph $G = (S, T, E)$, where the degree of every node $s \in S$ is exactly 2, and two sets $S_1, S_2 \subseteq S$ such that $S_1 \cup S_2 = S$. Finding a maximum-size subset $M \subseteq E$ for which both $M \cap E_1$ and $M \cap E_2$ are matchings, where E_i denotes the edges induced by T and S_i for $i = 1, 2$, is NP-hard. To reduce this problem to the problem in Theorem 5.1, we need an intermediate step. To this end, we construct a bipartite graph G', from which we can derive the temporal graph whose underlying graph is a tree.

Let $G' = (S, T', E')$ be a bipartite graph, where S is exactly the same set of nodes as in G. Let the set of nodes T have a fixed ordering $\{t_1, \dots, t_{|T|}\}$. For every $t_i \in T$, we add t_i^1, t_i^2, t_i^3, d_i to T', where d_i is a "dummy node". The edge set of G' is as follows. For every edge $st_i \in E$, add the edge st_i^1 to E' if $s \in S_1 \setminus S_2$, add the edge st_i^2 to E' if $s \in S_1 \cap S_2$ and add the edge st_i^3 to E' if $s \in S_2 \setminus S_1$. The degree of all dummy nodes is 0. We order the

nodes in T' as follows: $\{t'_1, \dots, t'_{4|T|}\} = \{t^1_1, t^2_1, t^3_1, d_1, \dots, t^1_{|T|}, t^2_{|T|}, t^3_{|T|}, d_{|T|}\}$. An example of the construction of G' is illustrated in Figure 2.

Now, we construct the temporal graph $\mathcal{G} = (\widehat{G}, \lambda)$, for which the corresponding underlying graph $\widehat{G}$ is a tree. The *core* of $\widehat{G}$ is a star, where the node v in the center has $|S|$ neighbors: $\{w_1, \dots, w_{|S|}\}$. The time-labels on the edges of the star is as follows. For every $i \in [|S|]$, let $\lambda(vw_i) = \{x, y\}$, provided that s_i is the neighbor of both t'_x and t'_y in G'. Since every node $s_i \in S$ has a degree of 2 in G', every edge of the star has exactly two time-labels.

We will get back to the construction of $\widehat{G}$, but first observe that if we solve the maximum Δ-matching problem on this star with $\Delta = 2$ and the constraint that at most one time-label can be selected for each edge, then we can solve the double matching problem on G. Indeed, we constructed G' and the previous star in such a way that there is a one-to-one correspondence between the solutions of the two problems. However, in Theorem 5.1, there is no such constraint that only one time-label can be chosen for each edge. In the following, we modify $\widehat{G}$ in such a way that there will always be an optimal solution for the maximum Δ-matching problem in which this constraint is satisfied.

Now, we continue the construction of $\widehat{G}$. We iterate through the nodes w_i for every $i \in [|S|]$, and connect new nodes to these to create an optimal solution for the maximum Δ-matching problem in which the above-mentioned constraint is satisfied. For every node w_i, we have three cases based on the time-labels on the edge vw_i. From now on, we assume that $\lambda(vw_i) = \{x, y\}$, where $x < y$. The three cases are as follows.

If $x + 1 = y$, the condition is automatically satisfied, since $\Delta = 2$. Therefore, we do not need extra edges to fix the construction.

If $x + 1 \neq y$ and $x \not\equiv y \bmod 2$, then we add $\frac{y-x-1}{2}$ new nodes to the graph, and denote them by $w^1_i, \dots, w^{(y-x-1)/2}_i$, and we also add the edges $w_i w^1_i, \dots, w_i w^{(y-x-1)/2}_i$. The time-labels on these edges are as follows. On the first edge, the time-labels are $\lambda(w_i w^1_i) = \{x+1, x+2\}$, and on the last edge, $\lambda(w_i w^{(y-x-1)/2}_i) = \{y-2, y-1\}$. We space the time-labels evenly, that is, $\lambda(w_i w^j_i) = \{x + 2j - 1, x + 2j\}$ for every $j \in [(y - x)/2]$. Now, we are done with the second case.

In the last case, we have $x + 1 \neq y$, where $x \equiv y \bmod 2$. First, we add one node $\widehat{w}_i$ to the tree, and the edge $w_i\widehat{w}_i$. The time-labels on this edge are $\lambda(w_i\widehat{w}_i) = \{x, y + 1\}$. From here, the construction is the same as before, since on the edge $w_i\widehat{w}_i$ the two labels are incongruent mod 2. Precisely, we add $\frac{y-x}{2}$ new nodes to the graph, named $\widehat{w}^1_i, \dots, \widehat{w}^{(y-x)/2}_i$, and we add the edges $\widehat{w}_i\widehat{w}^1_i, \dots, \widehat{w}_i\widehat{w}^{(y-x)/2}_i$. The labels on these edges are as follows. $\lambda(\widehat{w}_i\widehat{w}^j_i) = \{x + 2y - 1, x + 2y\}$ for every $j \in [(y - x)/2]$. With that, we have covered all cases and we are done with the construction of $\widehat{G}$, which is illustrated in Figure 3.

Now, consider the maximum Δ-matching problem on $\widehat{G}$ with the constraint that on the core star, we have to choose as many time edges as possible, but we can select at most one time-label per edge. We refer to this problem as the *core problem*.

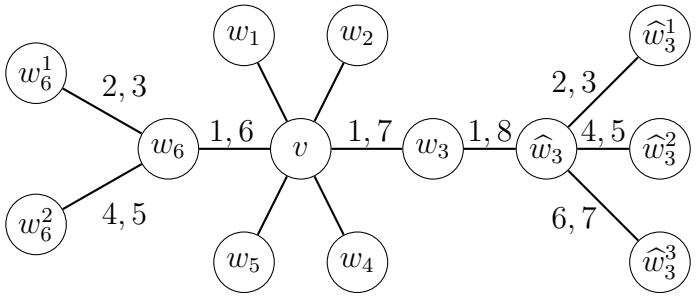


Figure 3: Example of the construction of $\widehat{G}$ in the proof of Theorem 5.1.

Remark 5.1. *If we can solve the core problem in polynomial time, we can also solve the double matching problem in polynomial time.*

Now, we prove two lemmas, which show that we can solve the core problem in polynomial time if we can solve the maximum Δ-matching problem on $\widehat{G}$ in polynomial time. Let us call a solution of the maximum Δ-matching problem on this tree *normalized* if the solution has at most one time-label on each edge of the core star.

Lemma 5.2. *For every optimal solution of the maximum Δ-matching problem on $\widehat{G}$, there exists an optimal normalized solution of the same size.*

Proof. Let M be an optimal solution to the maximum Δ-matching problem on $\widehat{G}$, where M contains both time edges of the edge vw_i of the core star. Observe that we have constructed $\widehat{G}$ in such a way that we can exclude either of the two time edges of vw_i and rearrange the selected time edges in the extra component connected to w_i, and still obtain a feasible solution to the maximum Δ-matching problem with the same size. Since the rearrangement of the chosen time edges in the extra component does not affect the rest of the tree, and by removing a chosen time-label from an edge of the core star we still obtain a feasible solution, we can repeat this process until we obtain a normalized solution. □

We have proven that there always exists a normalized optimal solution to the maximum Δ-matching problem in this tree. It remains to prove that such solutions, when restricted to the core star, are optimal solutions to the core problem.

Lemma 5.3. *Every optimal normalized solution is an optimal solution to the core problem.*

Proof. Observe that for every edge vw_i of the core star, the number of time edges that can be selected from the associated extra component is independent of whether we have selected zero or one time edges of vw_i. Let M be an optimal normalized solution to the maximum Δ-matching problem on $\widehat{G}$, which is not an optimal solution to the core problem. Let M_c denote the set of time edges of the core star, which are also in M. Since M is not an optimal solution to the core problem, there exists a solution M^*, which is also normalized, but $|M_c^*| > |M_c|$. Since both M and M^* are normalized, $M' = (M \setminus M_c) \cup M_c^*$ is a feasible

solution to the maximum Δ-matching problem, based on the previous observation. Since $|M_c^*| > |M_c|$, it follows that $|M'| > |M|$, which is a contradiction, since M is an optimal solution to the maximum Δ-matching problem. □

After finding an optimal solution to the maximum Δ-matching problem on $\widehat{G}$, we can construct a normalized solution based on Lemma 5.2 in polynomial time, from which we can obtain an optimal solution to the core problem in polynomial time. Therefore, if we could solve the maximum Δ-matching problem on $\widehat{G}$ in polynomial time, we could solve the double matching problem in polynomial time, based on Remark 5.1, which completes the proof of Theorem 5.1. □

With that, we have shown that the maximum Δ-matching problem is NP-hard, even on trees if every edge appears at most twice and $\Delta = 2$. From the proof of Theorem 5.1, it is clear that the problem remains NP-hard when $\Delta \geq 2$ and every edge appears at most twice, and also when $\Delta = 2$, but the edges can appear at least two times. But what can we say about approximation algorithms? Can we approximate the optimal solution to the maximum Δ-matching problem on trees arbitrarily well, or is this problem also APX-hard? In the Section 5.3, we address this question and present an Efficient Polynomial-Time Approximation Scheme. But first, let us present a solution to the problem, where the edges can appear at most once.

5.2 Single edge appearances

In Section 5.1, we showed that the maximum Δ-matching problem is NP-hard on trees even if the edges can appear at most twice. In this section, we prove that the problem can be solved in polynomial time if the edges can appear at most once.

Theorem 5.2. *The maximum Δ-matching problem can be solved in polynomial time if the underlying graph $G = (V, E)$ of the input temporal graph $\mathcal{G} = (G, \lambda)$ is a tree, and every edge appears at most once.*

Proof. We provide a dynamic programming solution to the problem. First, add a new node r to the set of nodes V, add a new edge rr' to the set of edges E, where $r' \in V$ is an arbitrary node, and let $\lambda(rr') = \emptyset$. Run a BFS (Breadth-First Search) algorithm from the node r to define parents, children, and rooted subtrees. Let us denote the parent of a node v by $p(v)$ and the subtree rooted at a node v by T_v. If a node v has children $v'_1, \ldots, v'_k$, then we call the edges $vv'_1, \ldots, vv'_k$ the *descendant edges* of v. Let $S_v = T_v \cup \{vp(v)\}$ denote the union of the subtree of v and the edge that connects v to its parent. Since the new node r does not have a parent, we define $p(r)$ and S_r as $p(r) = \varnothing,\ S_r = \varnothing$.

Now, we define the subproblems. For every node $v \in V \setminus \{r\}$, we define $\mathrm{OPT}(v, 0)$ as the maximum size of a Δ-matching in the subtree S_v that *does not include* the edge $vp(v)$

and we define $\mathrm{OPT}(v,1)$ as the maximum size of a Δ-matching in the subtree S_v that *includes* the edge $vp(v)$. Including or not including an edge means determining whether the given edge appears in our constructed Δ-matching with its assigned single time-label. Note that if an edge has no assigned time-label, it is never included.

Now, we demonstrate how to obtain the solution to the original problem and then we present the algorithm that solves the subproblems. We solve the problem starting from the leaves of the tree obtained during the BFS, and for each vertex v, we compute $\mathrm{OPT}(v,0)$ and $\mathrm{OPT}(v,1)$ using the optimal values computed for its children. The solution to the original problem (by definition) is $\mathrm{OPT}(r',0)$, where r' is the only neighbor of r.

Now, we present the algorithm that computes $\mathrm{OPT}(v,0)$ and $\mathrm{OPT}(v,1)$ for a given node v. Let $v'_1,\dots,v'_k$ be an ordering of the children of v, where $\lambda(vv'_1) \le \dots \le \lambda(vv'_k)$. If there is no time-label on an edge between v and a child of this node, then we do not include that child in this order. Now, we formulate two IP problems to obtain $\mathrm{OPT}(v,0)$ and $\mathrm{OPT}(v,1)$.

Let $c_i = \mathrm{OPT}(v'_i,1) - \mathrm{OPT}(v'_i,0)$ for every $i \in [k]$. The IP problem for $\mathrm{OPT}(v,0)$ is as follows.

$$\max \sum_{i\in[k]} x_i c_i \tag{5a}$$

s.t.

$$x \in \mathbb{Z}^k \tag{5b}$$

$$0 \le x_i \le 1 \qquad \forall i \in [k] \tag{5c}$$

$$\sum_{i=0}^{\Delta-1} x_{j+i} \le 1 \qquad \forall j \in [k-\Delta+1] \tag{5d}$$

We introduce a variable x_i for each child v'_i, which is either 1 or 0, depending on whether we include the edge leading to the child or not. The objective function means that we only consider how much we gain by including an edge in the solution, compared to the case where we do not include it. The only constraint is that we should not select two edges whose time-labels are closer to each other than Δ. This means, for the x variables, that if $i,j \in [k]$, $|i-j| < \Delta$, then either x_i or x_j (or both) is 0. Observe that the matrix of this IP problem is a network matrix and the right-hand side is a vector with only 1-s in it, hence the problem is solvable in polynomial time.

Now, we formulate the IP problem for $\mathrm{OPT}(v,1)$. The only change we need to make in the previous formulation is that now we cannot select time-labels from the Δ-neighborhood of $\lambda(vp(v))$. That is, we have to remove every v'_i child from $\{v'_1,\dots,v'_k\}$, where $|\lambda(vv'_i) - \lambda(vp(v))| < \Delta$. Everything else remains the same, so obviously, the matrix remains a network matrix, and the right-hand side also stays integer.

Now that we have shown how to compute a subproblem and demonstrated how to obtain the solution to the original problem, only the analysis of the runtime remains.

The BFS algorithm at the beginning runs in $O(n)$, where $|V| = n$ denotes the number of nodes. Computing $\mathrm{OPT}(v, 0)$ and $\mathrm{OPT}(v, 1)$ for a node v can be done in polynomial time, as shown before. Therefore calculating $\mathrm{OPT}(v, 0)$ and $\mathrm{OPT}(v, 1)$ for every node v requires polynomial time, hence we proved the theorem. □

5.3 An Efficient Polynomial-Time Approximation Scheme

We know that there is no Polynomial-Time Approximation Scheme (PTAS) for the maximum Δ-matching problem for any $\Delta \geq 2$ and $\mathcal{T} \geq 3$, unless P=NP [9]. Now, we will show that the problem admits an EPTAS if the underlying graph of the input temporal graph is a tree and Δ is constant. Our $(1-\varepsilon)$-approximation algorithm uses some simple ideas from the $\frac{\Delta}{2\Delta-1}$-approximation algorithm presented in [9], but we take advantage of the fact that our graph is a tree and hence we can solve harder problems with dynamic programming. Throughout this section, we assume that $0 < \Delta$, since the $\Delta = 0$ case is trivial.

The main idea of our algorithm is to compute maximum matchings for time slices of size k within the input temporal graph, where k is a constant. These slices are chosen so that they are sufficiently spaced apart, ensuring that they do not interfere with each other. This allows the matchings to be computed in polynomial time. After identifying these non-overlapping slices, we greedily fill in the remaining gaps. We systematically explore different combinations of non-interfering slices of size k and select the largest Δ-matching found. Using counting arguments, we establish that this approach achieves the desired approximation ratio.

Now we introduce additional notation and terminology, closely following that of [9]. Let k and $\mathcal{T}$ be natural numbers with $\Delta \leq k \leq \mathcal{T}$. For each time tick $\tau \in [\mathcal{T} - k + 1]$, we define the *$k$-window* W_τ as the interval $[\tau, \tau + k - 1]$ of length k. An interval of length at most $k-1$ that either begins at tick 1 or ends at tick $\mathcal{T}$ is referred to as a *partial k-window (with respect to lifetime $\mathcal{T}$)*. For brevity, we simply use the term *partial k-window* in this problem. The *distance* between two disjoint intervals $[a_1, b_1]$ and $[a_2, b_2]$ with $b_1 < a_2$ is defined as $b_1 - a_2$.

A *k-template (with respect to lifetime $\mathcal{T}$)* is a family of maximal cardinality $\mathcal{S}$ of k-windows or partial k-windows, such that any two consecutive intervals in $\mathcal{S}$ are at distance exactly $\Delta - 1$ from each other. A Δ-matching $M^{\mathcal{S}}$ in $\mathcal{G} = (G, \lambda)$ is called a Δ-matching *with respect to k-template $\mathcal{S}$* if $M^{\mathcal{S}}$ has the maximum possible number of edges in every interval $W \in \mathcal{S}$.

We say that a time tick τ is *covered* by a k-template $\mathcal{S}$ if τ belongs to an interval of $\mathcal{S}$. First, we present some properties of k-templates which we need to prove the approximation

ratio of our algorithm, then we will show that we can solve the maximum Δ-matching problem restricted to a k-template in polynomial time. The following is a rephrasing of Lemma 24 in [9] for our situation.

Lemma 5.4. *Let k and $\mathcal{T}$ be natural numbers such that $\Delta \leq k \leq \mathcal{T}$. Then*

(1) there are exactly $k + \Delta - 1$ different k-templates with respect to lifetime $\mathcal{T}$, and

(2) every time tick in $[\mathcal{T}]$ is covered by exactly k distinct k-templates.

Proof. We start by proving (1). Observe that a k-template is uniquely determined by its leftmost interval. Indeed, by fixing the leftmost interval, the positions of the following intervals are determined. Also observe that the leftmost interval is either a partial k-window that starts at time tick 1, or a (possibly partial) k-window that starts in one of the first Δ time ticks. Since there are $k - 1$ intervals of the first type and Δ intervals of the second type, we conclude that there are exactly $k + \Delta - 1$ k-templates with respect to lifetime $\mathcal{T}$.

To prove (2), observe that all k-templates can be derived step by step from a starting k-template $\mathcal{S}$, where the first interval is the first tick, which is a partial k-window. Indeed, we can shift all intervals of the current k-template one time tick to the right in each step. It follows that each time slot is covered exactly k times across the $k + \Delta - 1$ shifting iterations. □

By definition, for any two distinct intervals W_1 and W_2 in a k-template $\mathcal{S}$ and for any two time ticks $\tau_1 \in W_1$ and $\tau_2 \in W_2$ we have $|\tau_1 - \tau_2| > \Delta$, which implies that no two time edges of $\mathcal{G}$ that appear in time slots of different intervals of $\mathcal{S}$ are in conflict. This observation implies that finding a Δ-matching with respect to $\mathcal{S}$ can be reduced to computing a maximum Δ-matching in $\mathcal{G}|_W$ for every $W \in \mathcal{S}$. All that remains is to show that we can solve the maximum Δ-matching problem in polynomial time on an interval of length k, where k is a constant. This brings us to our second lemma, which is as follows.

Lemma 5.5. *Maximum Δ-matching can be solved in polynomial time if the underlying graph $G = (V, E)$ of the input temporal graph $\mathcal{G} = (G, \lambda)$ is a tree, and the lifetime $\mathcal{T}$ is constant.*

Proof. We provide a dynamic programming solution to the problem, which is similar to the solution used in the proof of Theorem 5.2. Similar to the previous proof, add a new node r to the set of nodes V, add a new edge rr' to the set of edges E, where $r' \in V$ is an arbitrary node, and let $\lambda(rr') = \emptyset$. Run a BFS (Breadth-First Search) algorithm from the node r to define parents, children, and rooted subtrees. Let us denote the parent of a node v by $p(v)$ and the subtree rooted at a node v by T_v. If a node v has children $v'_1, \ldots, v'_k$, then we call the edges $vv'_1, \ldots, vv'_k$ the *descendant edges* of v. Let $S_v = T_v \cup \{vp(v)\}$

denote the union of the subtree of v and the edge that connects v to its parent. Since the new node r does not have a parent, we define $p(r)$ and S_r as $p(r) = \varnothing,\ S_r = \varnothing$.

Now, we define the subproblems. For every node $v \in V \setminus \{r\}$ and every subset $I \subseteq \lambda(vp(v))$, we define $\mathrm{OPT}(v, I)$ as the maximum size of a Δ-matching in the subtree S_v that *includes* the edge $vp(v)$ with the time-labels in I. Including an edge with a subset of time-labels means that the given edge appears in our constructed Δ-matching M with those and only those time-labels.

Now, we demonstrate how to obtain the solution to the original problem and then we present the algorithm that solves the subproblems. We solve the problem starting from the leaves of the tree obtained during the BFS, and for each vertex v, we compute $\mathrm{OPT}(v, I)$ for all $I \in \lambda(vp(v))$ using the optimal values computed for its children. The solution to the original problem (by definition) is $\mathrm{OPT}(r', \emptyset)$, where r' is the only neighbor of r.

Now, we present the algorithm that computes $\mathrm{OPT}(v, I)$. Let $P(v, I) \subseteq [\mathcal{T}]$ be the subset with maximum cardinality such that for every $j \in P(v, I)$ and every $i \in I,\ |i-j| \geq \Delta$ holds. That is, $P(v, I)$ is the set of the possible time-labels on the descendant edges of v, provided that $vp(v)$ is included in the solution with the time-labels in I. We choose a subset $L \subseteq P(v, I)$ and a partition $\{L_1, \ldots, L_h\}$ of L, such that for every set L_i in this partition, there will be a descendant edge that is included into the solution with the time-labels in L_i. Moreover, every descendant edge must be included into the solution with either the empty set or with the time-labels in one of the subsets L_i.

We enumerate all partitions and — for each of them — determine which set corresponds to which edge by solving a matching problem, which we call the *pairing phase*. By solving these matching problems, we obtain lower bounds to $\mathrm{OPT}(v, I)$. But we do not merely aim to provide a lower bound — we seek to precisely determine the optimal value, and therefore we examine all possibilities by selecting every possible subset $L \subseteq P(v, I)$ and considering all possible partitions of L. This means that after the pairing phase we move on to a new partition of the currently examined $L \subseteq P(v, I)$, and after we check every partition, we move on to the next subset $L' \subseteq P(v, I)$.

Some partitions yield "trivially invalid" results, therefore we skip the pairing phase for these partitions and move on to the next one. These partitions are as follows. First, if the number of sets in the partition is greater than the number of descendant edges, then we skip the pairing phase. Second, if there are two time ticks $x, y \in L$ for which $|x - y| < \Delta$ is true, we also skip the pairing phase.

Now we provide a precise description of the pairing phase. At the beginning of this phase, we have $L \subseteq P(v, I)$ and a partition $\{L_1, \ldots, L_h\}$ of L. Let $\{e_1, \ldots, e_k\}$ be an arbitrary ordering of the descendant edges of v, where e_i denotes the edge vv_i'. We find a maximum-weight matching on the bipartite graph $G' = (S', T', E')$, that covers the entire vertex set S', where the nodes in S' correspond to the sets in the partition, and the nodes

in T' correspond to the descendant edges. Let $s_j \in S'$ denote the vertex corresponding to L_j, and let $t_i \in T'$ denote the vertex corresponding to e_i. Let $s_j t_i$ be an edge in E' if and only if $L_j \subseteq \lambda(e_i)$. We also define a weight function $w' : E' \to \mathbb{R}$ on the edges. For every $s_j t_i \in E'$, let $w'(s_j t_i) = \mathrm{OPT}(v'_i, L_j)$.

In the pairing phase, we find a maximum-weight matching in G' that covers S' (if there is no such matching for the current partition, we move on to the next one). We then assign the sets in the partition to the descendant edges according to the edges of this matching. If we did not assign anything to a descendant edge, then we include that edge into the solution with the empty set.

Note that with this assignment, we get a feasible solution on S_v. Therefore, we can obtain a lower bound to $\mathrm{OPT}(v, I)$ by adding up the $\mathrm{OPT}(w_i, I_{w_i})$ values, where the set I_{w_i} was assigned to w_i, and also adding $|I|$ to this sum, since we are including the edge $vp(v)$ with the time-labels in I. Since we iterate through every $L \subseteq P(v, I)$ and every partitions of L, we will find the subset and the partition which correspond to the optimal solution $\mathrm{OPT}(v, I)$.

Now that we have shown how to compute a subproblem and demonstrated how to obtain the solution to the original problem, only the analysis of the runtime remains.

The BFS algorithm at the beginning runs in $O(n)$, where $|V| = n$ denotes the number of nodes. Computing $\mathrm{OPT}(v, I)$ for a node v and a set $I \subseteq [\mathcal{T}]$ requires iterating through all $O(2^{\mathcal{T}})$ subsets of $P \subseteq [\mathcal{T}]$, and all $O(\mathcal{T}^{\mathcal{T}})$ partitions of the chosen subset. This is a total of $O((2\mathcal{T})^{\mathcal{T}})$ iterations. In (possibly) every iteration we have to solve a maximum-weight matching problem on a bipartite graph, where we have to cover one of the vertex sets, which has a constant number of elements. Let us denote the number of nodes in the constant-sized vertex set by c. Observe that we can solve this problem in $O(n + \mathcal{T}^4)$ time by selecting the c incident edges with the largest weights for every node in the constant-sized vertex set and then using the Hungarian method [21] only for these edges. We can select these edges in $O(n)$ time and the Hungarian method runs in $O(c^4) = O(\mathcal{T}^4)$ time in this simplified graph. To compute $\mathrm{OPT}(v, I)$ for a node v and every subset $I \subseteq [\mathcal{T}]$, we have to repeat the same process $O(2^{\mathcal{T}})$ times, which results in another $O(2^{\mathcal{T}} \cdot (2\mathcal{T})^{\mathcal{T}} \cdot (n + \mathcal{T}^4)) = O(n(4\mathcal{T})^{\mathcal{T}} + 4^{\mathcal{T}} + \mathcal{T}^{\mathcal{T}+4}) = O(n(4\mathcal{T})^{\mathcal{T}})$ extra runtime. Therefore calculating $\mathrm{OPT}(v, I)$ for every node v and every subset I requires $O(n^2(4\mathcal{T})^{\mathcal{T}})$ time. □

Given a k-template $\mathcal{S}$, every interval $W \in \mathcal{S}$ has at most k consecutive time ticks in it. Therefore, by relabeling the time ticks (subtracting the smallest value from every tick), we arrive to the same problem as in Lemma 5.5.

Corollary 5.1. *On an interval W of length k, where k is a constant, the maximum Δ-matching problem can be solved in polynomial time.*

We are now ready to present the approximation algorithm that we aim to prove is an EPTAS. Since our algorithm will be a $(1 - \varepsilon)$-approximation, we can assume that $\varepsilon < 1$.

The algorithm itself is simple: take $k \in \mathbb{Z}_+$ such that $k \leq \Delta/\varepsilon$, $\frac{(1-\varepsilon)(\Delta-1)}{\varepsilon} \leq k$, and $\Delta \leq k$ (it is easy to see that such k always exists), and for every k-template $\mathcal{S}$, compute a Δ-matching $M^{\mathcal{S}}$ with respect to $\mathcal{S}$ and return one of maximum cardinality. Now we prove that this algorithm is indeed an Efficient Polynomial-Time Approximation Scheme.

Theorem 5.3. *For every $0 < \varepsilon < 1$, the maximum Δ-matching problem admits an $O(\mathcal{T}n^2(4\Delta/\varepsilon)^{\Delta/\varepsilon})$-time $(1-\varepsilon)$-approximation algorithm if the underlying graph G of the input temporal graph $\mathcal{G}$ is a tree and $\Delta \in \mathbb{Z}_+$ is a positive constant.*

Proof. Let $\mathcal{G} = (G, \lambda)$ be an arbitrary temporal graph of lifetime $\mathcal{T}$ and Δ be a positive constant such that $\Delta \leq \mathcal{T}$. Let M^* be an optimal solution to the maximum Δ-matching problem on $\mathcal{G}$.

Clearly, the algorithm outputs a feasible solution as the output M is a Δ-matching with respect to some k-template. We show next that M is the desired approximate solution. Let $\mathcal{C}$ be the family of all k-templates with respect to lifetime $\mathcal{T}$, and let $\mathcal{S}' \in \mathcal{C}$ be a k-template such that $M = M^{\mathcal{S}'}$. It follows from the algorithm that $|M^{\mathcal{S}'}| \geq |M^{\mathcal{S}}|$ for every $\mathcal{S} \in \mathcal{C}$. By definition, for every $\mathcal{S} \in \mathcal{C}$ and for every interval $W \in \mathcal{S}$, we have $\sum_{\tau \in W} |M^{\mathcal{S}}_\tau| \geq \sum_{\tau \in W} |M^*_\tau|$, where $M_\tau = M \cap E_\tau$. Therefore

$$|M^{\mathcal{S}}| = \sum_{W \in \mathcal{S}} \sum_{\tau \in W} |M^{\mathcal{S}}_\tau| \geq \sum_{W \in \mathcal{S}} \sum_{\tau \in W} |M^*_\tau|. \tag{6}$$

Using the above inequalities and Lemma 5.4, we derive

$$(k+\Delta-1)|M^{\mathcal{S}'}| \geq \sum_{\mathcal{S} \in \mathcal{C}} |M^{\mathcal{S}}| \geq \sum_{\mathcal{S} \in \mathcal{C}} \sum_{W \in \mathcal{S}} \sum_{\tau \in W} |M^{\mathcal{S}}_\tau| \geq \sum_{\mathcal{S} \in \mathcal{C}} \sum_{W \in \mathcal{S}} \sum_{\tau \in W} |M^*_\tau| = k \sum_{\tau=1}^{\mathcal{T}} |M^*_\tau| = k|M^*|. \tag{7}$$

Since we chose k such that $\frac{(1-\varepsilon)(\Delta-1)}{\varepsilon} \leq k$, we have $\frac{\Delta-1}{k+\Delta-1} \leq \varepsilon$, hence

$$|M| = |M^{\mathcal{S}'}| \geq \frac{k}{k+\Delta-1}|M^*| = (1 - \frac{\Delta-1}{k+\Delta-1})|M^*| \geq (1-\varepsilon)|M^*|. \tag{8}$$

Now we analyze the time complexity of the algorithm. By Lemma 5.4, there are exactly $k+\Delta-1$ different k-templates, and since every interval W in a k-template $\mathcal{S}$ has a length at most k, there are $O(\mathcal{T}/k)$ intervals in a k-template. By Corollary 5.1, on an interval of length k, we can solve the restricted problem in $O(n^2(4k)^k)$ time. Therefore, the total runtime of our algorithm is $O(\mathcal{T}n^2(4k)^k)$. Since we chose k such that $k \leq \Delta/\varepsilon$, in terms of ε, the runtime is $O(\mathcal{T}n^2(4\Delta/\varepsilon)^{\Delta/\varepsilon})$, hence our algorithm is indeed an EPTAS. □

From the proofs of Lemma 5.5 and Theorem 5.3, it is clear that with a small modification, the algorithm works for the weighted case as well.

Remark 5.2. *For every $0 < \varepsilon < 1$, the weighted maximum Δ-matching problem admits an $O(\mathcal{T}n^2(4\Delta/\varepsilon)^{\Delta/\varepsilon})$-time $(1-\varepsilon)$-approximation algorithm if the underlying graph G of the input temporal graph $\mathcal{G}$ is a tree and $\Delta \in \mathbb{Z}_+$ is a positive constant.*

6 Open questions

We have shown that among the Δ-, γ-, and d-matchings, certain pairs can be reduced to one another. However, many potential connections remain unexplored, and we have not proven in any case that one problem cannot be reduced to another. This raises the question of whether we have uncovered all possible relationships between these problems, or whether there might still be reductions that remain to be discovered. If the non-bipartite d-matching problem cannot be reduced to the other problems, another important question arises: is this problem also APX-hard, or can it be approximated arbitrarily well?

Concerning the inverse problems of the maximum Δ-matching, it remains open whether Problem 4.2 is NP-hard if $c < \lfloor k/\Delta \rfloor$ and $\Delta > 0$. Furthermore, both in the case of the weighted inverse problem of the maximum Δ-matching (Problem 4.5) and the bagpiper edge-coloring problem (Problem 4.9), it remains open whether the problem is NP-hard when only the edges have weights and the input graph is bipartite.

References

[1] Shubham Gupta and Srikanta Bedathur. *A Survey on Temporal Graph Representation Learning and Generative Modeling*. 2022. arXiv: 2208.12126.

[2] Othon Michail. "An Introduction to Temporal Graphs: An Algorithmic Perspective". In: *Algorithms, Probability, Networks, and Games: Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday*. Ed. by Christos Zaroliagis, Grammati Pantziou, and Spyros Kontogiannis. Cham: Springer International Publishing, 2015, pp. 308–343.

[3] Mohammad Mehdi Hosseinzadeh, Mario Cannataro, Pietro Hiram Guzzi, and Riccardo Dondi. "Temporal networks in biology and medicine: a survey on models, algorithms, and tools". In: *Network Modeling Analysis in Health Informatics and Bioinformatics* 12.1 (2022), p. 10. ISSN: 2192-6670.

[4] Pete Austin, Sougata Bose, and Patrick Totzke. "Parity Games on Temporal Graphs". In: *Foundations of Software Science and Computation Structures*. Ed. by Naoki Kobayashi and James Worrell. Cham: Springer Nature Switzerland, 2024, pp. 79–98. ISBN: 978-3-031-57228-9.

[5] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. "Temporal vertex cover with a sliding time window". In: *Journal of Computer and System Sciences* 107 (2020), pp. 108–123. ISSN: 0022-0000.

[6] Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. "Interference-free walks in time: temporally disjoint paths". In: *Autonomous Agents and Multi-Agent Systems* 37.1 (2022), p. 1. ISSN: 1573-7454.

[7] Markus Chimani, Niklas Troost, and Tilo Wiedera. "Approximating Multistage Matching Problems". In: *Algorithmica* 84.8 (2022), pp. 2135–2153. ISSN: 1432-0541.

[8] Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. "Multistage Matchings". In: *Leibniz International Proceedings in Informatics (LIPIcs)*. Vol. 101. Leibniz International Proceedings in Informatics (LIPIcs). Malmo, Sweden, 2018, 7:1–7:13.

[9] George B Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. "Computing maximum matchings in temporal graphs". In: *Journal of Computer and System Sciences* 137 (2023), pp. 1–19.

[10] Julien Baste, Binh-Minh Bui-Xuan, and Antoine Roux. "Temporal matching". In: *Theoretical Computer Science* 806 (2020), pp. 184–196.

[11] Péter Madarasi. "Matchings under distance constraints I." In: *Annals of Operations Research* 305 (2021), pp. 137–161.

[12] Péter Madarasi. "Matchings under distance constraints II." In: *Annals of Operations Research* 332 (2024), pp. 303–327.

[13] David Kempe, Jon Kleinberg, and Amit Kumar. "Connnectivity and Inference Problems for Temporal Networks". In: *Journal of Computer and System Sciences* 64 (2002), pp. 820–842.

[14] Philipp Zschoche. "A faster parameterized algorithm for temporal matching". In: *Information Processing Letters* 174 (2022), p. 106181.

[15] Timothe Picavet, Ngoc-Trung Nguyen, and Binh-Minh Bui-Xuan. "Temporal Matching on Geometric Graph Data". In: *Algorithms and Complexity*. Ed. by Tiziana Calamoneri and Federico Corò. Cham: Springer International Publishing, 2021, pp. 394–408.

[16] Ian Holyer. "The NP-completeness of edge-coloring". In: *SIAM J. Comp.* 10(4) (1981), pp. 718–720.

[17] Mkrtchyan Vahan. "The maximum 2-edge-colorable subgraph problem and its fixed-parameter tractability". In: *Journal of Graph Algorithms and Applications* 28(1) (2024), pp. 129–147.

[18] Mkrtchyan Vahan. *Two hardness results for the maximum 2-edge-colorable subgraph problem in bipartite graphs*. Sept. 2024. arXiv: 2409.15388.

[19] Bengt Aspvall and Richard E Stone. "Khachiyan's linear programming algorithm". In: *Journal of Algorithms* 1.1 (1980), pp. 1–13. ISSN: 0196-6774.

[20] A. J. Hoffman and J. B. Kruskal. "Integral Boundary Points of Convex Polyhedra". In: *Linear Inequalities and Related Systems*. Ed. by H. W. Kuhn and A. W. Tucker. Vol. 38. Annals of Mathematics Studies. Princeton University Press, 1956, pp. 223–246.

[21] Harold W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97.

MI alapú eszközök használatáról szóló nyilatkozat

Alulírott Juhász Márk Hunor nyilatkozom, hogy szakdolgozatom elkészítése során az alább felsorolt feladatok elvégzésére a megadott MI alapú eszközöket alkalmaztam:

Feladat	**Felhasznált eszköz**	**Felhasználás helye**	**Megjegyzés**
Általam írt magyar mondatok angolra fordítása	ChatGPT	Teljes dolgozat	Tisztán ChatGPT-vel generált mondat nem szerepel a dolgozatban

A felsoroltakon túl más MI alapú eszközt nem használtam.

Dátum: 2025. 06. 01.