Chain-Constrained Spanning Trees*

Neil Olver† and Rico Zenklusen‡

January 30, 2020

Abstract

We consider the problem of finding a spanning tree satisfying a family of additional constraints. Several settings have been considered previously, the most famous being the problem of finding a spanning tree with degree constraints. Since the problem is hard, the goal is typically to find a spanning tree that violates the constraints as little as possible.

Iterative rounding has become the tool of choice for constrained spanning tree problems. However, iterative rounding approaches are very hard to adapt to settings where an edge can be part of more than a constant number of constraints. We consider a natural constrained spanning tree problem of this type, namely where upper bounds are imposed on a family of cuts forming a chain. Our approach reduces the problem to a family of independent matroid intersection problems, leading to a spanning tree that violates each constraint by a factor of at most 9.

We also present strong hardness results: among other implications, these are the first to show, in the setting of a basic constrained spanning tree problem, a qualitative difference between what can be achieved when allowing multiplicative as opposed to additive constraint violations. **Keywords:** network design spanning trees approximation algorithms

1 Introduction

Spanning tree problems with additional $\{0,1\}$-packing constraints have spawned considerable interest recently. This development was motivated by a variety of applications, including VLSI design, vehicle routing, and applications in communication networks [**?**, **?**, **?**]. Since even finding a feasible solution of a constrained spanning tree problem is typically NP-hard, the focus is on efficient procedures that either certify that the problem has no feasible solution, or find a spanning tree that violates the additional constraints by as little as possible. Often, an objective function to be minimized is also provided; here, however, we focus just on minimizing the constraint violations.

A wide variety of constrained spanning tree problems have been studied. Unfortunately, for most settings, little is known about what violation of the constraints must be accepted in order that a solution can be efficiently obtained. An exception is the most classical problem in this context, the degree-bounded spanning tree problem. Here the goal is to find a spanning tree $T \subseteq E$ in a graph $G = (V, E)$ such that T satisfies a degree bound for each vertex $v \in V$, i.e., $|\delta(v) \cap T| \leq b_v$. For this problem, Fürer and Raghavachari [**?**] presented an essentially best possible algorithm that either shows that no spanning tree satisfying the degree constraints

*This project was supported by NSF grant CCF-1115849, an NWO Veni grant, and Swiss National Science Foundation Grant 200021_165866.

†Vrije Universiteit Amsterdam, The Netherlands n.olver@vu.nl and CWI, The Netherlands olver@cwi.nl.

‡ETH Zurich, Switzerland ricoz@math.ethz.ch.

exists, or returns a spanning tree violating each degree constraint by at most 1. We call this an *additive* 1-*approximation*, in contrast to an α-approximation, where each constraint can be violated by a *factor* $\alpha > 1$.

Recently, iterative rounding/relaxation algorithms became the tool of choice for dealing with constrained spanning tree problems. A cornerstone for this development was the work of Singh and Lau [**?**], which extended the iterative rounding framework of Jain [**?**] with a relaxation step. They obtained an additive 1-approximation even for the *minimum* degree-bounded spanning tree problem, i.e., the cost of the tree they return is upper bounded by the cost of an optimal solution not violating any constraints. This result was the culmination of a long sequence of papers presenting methods with various trade-offs between constraint violation and cost (see [**?**, **?**, **?**, **?**, **?**] and references therein).

Singh and Lau's iterative relaxation technique was later generalized by Bansal, Khandekar and Nagarajan [**?**], to show that even when upper bounds are given on an arbitrary family of edge sets $E_1, \ldots, E_k$, one can still find a (minimum cost) spanning tree violating each constraint by not more than $\max_{e \in E} |\{i \in [k] \mid e \in E_i\}| - 1$. If each edge is only in a constant number of constraints, this leads to an additive $O(1)$-approximation. But extending the iterative rounding technique beyond such settings seems to typically be very difficult. Some progress was achieved by Bansal, Khandekar, Könemann, Nagarajan and Peis [**?**], who used an iterative approach that iteratively replaces constraints by weaker ones, leading to an additive $O(\log n)$-approximation if the constraints are upper bounds on a laminar family of cuts. They left open whether an additive or multiplicative $O(1)$-approximation is possible in this setting, even when the cuts form a chain. Recently, Zenklusen [**?**] presented an additive $O(1)$-approximation for generalized degree bounds, where for each vertex an arbitrary matroid constraint on its adjacent edges has to be satisfied. This algorithms differs from previous iterative rounding approaches in that it successively simplifies the problem to reach a matroid intersection problem, rather than attempting to eliminate constraints until only spanning tree constraints remain.

To the best of our knowledge, with the exception of the setting of Zenklusen [**?**], no $O(1)$-approximations are known for constrained spanning tree problems where an edge can lie in a super-constant number of (linear) constraints. This seems to be an important difficulty that current techniques have trouble overcoming. Furthermore, in many settings, it is not well understood if finding an additive approximation is any harder than a multiplicative one. In particular, no constrained spanning tree problem was previously known where an $O(1)$-approximation is possible, but an additive $O(1)$-approximation is not. The goal of this paper is to address these points by studying chain-constrained spanning trees—a natural constrained spanning tree problem that evades current techniques.

1.1 Our results

The chain-constrained spanning tree problem is the following. We are given an undirected graph $G = (V, E)$ together with a family of cuts $\emptyset \subsetneq S_1 \subsetneq S_2, \ldots, \subsetneq S_\ell \subsetneq V$ forming a chain, and bounds $b_1, \ldots, b_\ell \in \mathbb{Z}_{>0}$. The goal is to find a spanning tree T that satisfies all of the constraints, i.e.,

$$|T \cap \delta(S_i)| \leq b_i \quad \forall i \in [\ell], \tag{1}$$

if such a spanning tree exists. Here, $\delta(S_i)$ denotes the set of edges with precisely one endpoint in S_i.

Notice that chain constraints allow edges to be in a super-constant number of constraints. It is also a natural candidate problem that captures many of the difficulties faced when trying to construct $O(1)$-approximations for the laminar case.

Our main algorithmic result is the following.

Theorem 1.1. *There is an efficient* 9*-approximation for the chain-constrained spanning tree problem.*

Like most work in the area, we exploit the natural LP relaxation of the problem. This relaxation asks for a point x in the spanning tree polytope P_{ST} which satisfies the constraints. But our method is not based on iterative rounding, which has become the standard tool. Instead, we reduce the problem to a family of independent matroid intersection problems. In order to do this, we decompose the problem into a number of independent subproblems, based on the laminar decomposition induced by a maximal family of independent tight spanning tree constraints of the solution to the LP relaxation. By a judicious choice of objective function, we are able to ensure that each of the resulting subproblems has a convenient structural property, namely, they have no *rainbows* in their support. A rainbow is a pair of edges e, f such that e is in a proper superset of the chain constraints in which f is contained. Within a subproblem, the lack of rainbows yields a natural "left-to-right" ordering of the edges in its support. This ordering is crucially exploited in order to define a partition matroid with the property that any independent set of this matroid does not contribute much more to any constraint then the fractional solution for the subproblem.

Even though the high-level approach is quite clean, there are several difficulties we have to address. In particular, it turns out to be impossible to obtain a multiplicative guarantee within each subproblem separately. Instead we must use a more relaxed target for each subproblem that nevertheless yields a constant multiplicative guarantee overall.

It is interesting to compare our approach to the one taken by Goemans [**?**] in work giving an additive 2-approximation to the minimum degree-bounded spanning tree problem with no loss in cost. Like our result, this result is not based on iterative rounding. Instead, local sparsity of an extreme point solution is exploited to argue that the edges of G can be oriented so that each node has indegree at most 2. The degree bound at a vertex v is then relaxed to involve only the edges which are oriented away from v, so that each edge occurs in only one constraint; the degree bounds are thus replaced by a single partition matroid. The result then follows by applying matroid intersection.

Our approach can be seen in a somewhat similar light. Using the rainbow-free structure in the subproblems, we eventually end up with a partition matroid in each subproblem. One can of course combine these resulting partition matroids over all the subproblems, to obtain a single global partition matroid. So at a high level, both algorithms proceed by relaxing (or in our case, replacing) the given constraints by a matroid constraint, and then applying matroid intersection.

We complement our $O(1)$-approximation result by showing that an additive $O(1)$-approximation is impossible (assuming P $\neq$ NP). As mentioned, this is the first result showing such a separation between what can be achieved additively and multiplicatively for a constrained spanning tree problem. Let n denote the number of vertices of G.

Theorem 1.2. *For the chain-constrained spanning tree problem it is NP-hard to distinguish between the cases where a spanning tree satisfying the chain constraints exists, and the case that every spanning tree violates some degree bound by* $\Omega(\log n/\log\log n)$ *units.*

This result is given in Section 3. Previously, the only hardness result of a similar nature was presented by Bansal et al. [**?**] for a very general constrained spanning tree problem, where constraints $|T \cap E_i| \leq b_i \ \forall i \in [k]$ are given for an *arbitrary* family of edge sets $E_1, \ldots, E_k \subseteq E$. They showed that unless NP has quasi-polynomial time algorithms, there is no additive $(\log^c n)$-approximation for this case, for some small constant $c \in (0, 1)$. Notice that our hardness result is stronger in terms of the approximation ratio, the underlying constrained spanning tree

model, and the complexity assumption. Furthermore, Theorem 1.2 shows that the additive $O(\log n)$-approximation of Bansal et al. [**?**] for the laminar-constrained spanning tree problem is close to optimal.

1.2 Thin trees

Given a graph G and a point x in the spanning tree polytope of G, a spanning tree T is called α-*thin* with respect to x if

$$|T \cap \delta(S)| \leq \alpha \cdot x(\delta(S)) \qquad \forall S \subseteq V.$$

The problem of finding an α-thin tree can be interpreted as a constrained spanning tree problem, where an upper bound $b_S := x(\delta(S))$ is imposed on every cut $\delta(S)$ of the graph. By construction, this exponentially sized LP has a feasible solution, and an α-approximate solution is exactly an α-thin spanning tree.

The concept of thin spanning trees gained considerably in relevance when Asadpour, Goemans, Madry, Oveis Gharan and Saberi [**?**] showed that an efficient algorithm for finding an α-thin spanning tree leads to an $O(\alpha)$-approximation for the Asymmetric Traveling Salesman Problem (ATSP)[1]. In a very recent tour de force, Anari and Oveis Gharan [**?**] gave a nonconstructive proof of the existence of polyloglog n-thin trees, which implies the same bound on the integrality gap of ATSP. The best constructive result for thin spanning trees (and ATSP) yield $O(\log n / \log\log n)$-thin spanning trees [**?**, **?**]. It is open whether $O(1)$-thin spanning trees exist, which (if shown constructively) would immediately imply an $O(1)$-factor approximation for ATSP. The chain-constrained spanning tree problem (as well as other variants where constraints are placed on only some cuts) can be seen as an easier variant of the thin tree problem. Our work can be seen as a small step towards an attack on the thin tree conjecture using combinatorial methods.

2 The algorithm

To simplify the exposition, we assume that we are dealing with a maximal chain of constraints imposed on the spanning tree. So we may label the vertices $V = \{v_1, \ldots, v_n\}$ of G such that $S_i = \{v_1, \cdots, v_i\}\ \forall\ i \in [n-1]$, the constraints being $|T \cap \delta(S_i)| \leq b_i$ for all $i \in [n-1]$. This is clearly not restrictive since by choosing a large right-hand side, any constraint can be made redundant.

Recall that the natural LP for the problem asks for a point x in the spanning tree polytope P_{ST} of G satisfying $x(\delta(S_i)) \leq b_i$ for all $i \in [n-1]$. Our algorithm starts with a fractional solution of this relaxation (if the relaxation is infeasible, this provides a certificate that the given instance has no solution). But we do not begin with an arbitrary feasible solution; we require an optimal solution with respect to a carefully chosen objective. More precisely, we take a solution that minimizes the total length of the edges, where the length of an edge $\{v_i, v_j\} \in E$ is $|i-j|$. Equivalently, the length of an edge is the number of chain constraints to which the edge contributes. This leads to the LP (2) shown below. Let x^* be an optimal solution to (2), which can be computed by standard techniques (see [**?**]). Notice that the objective function of (2) is the same as the total load on all cuts: $\sum_{i=1}^{n-1} x(\delta(S_i))$.

[1] Strictly speaking, Asadpour et al.'s approach required the spanning tree not only to be thin, but also to be of low cost. However this second requirement is not necessary for the mentioned statement to be true (see [**?**]).

$$\begin{aligned} \min \sum_{\{v_i,v_j\}\in E} |i-j|\cdot x(\{v_i,v_j\}) & \\ x &\in P_{ST} \\ x(\delta(S_i)) &\leq b_i \qquad\qquad \forall\, i\in[n-1] \end{aligned} \tag{2}$$

The above objective function is motivated by a subprocedure we use to find a spanning tree in an instance that does not contain what we call a *rainbow*. A rainbow consists of two edges $\{v_i,v_j\},\{v_p,v_q\}$ with $i\leq p<q\leq j$ and either $i<p$ or $q<j$, i.e., the first edge is in a proper superset of the chain constraints in which the second edge is in. Even though the above objective function does not necessarily lead to an LP solution x^* whose support $\mathrm{supp}(x^*)=\{e\in E\mid x^*(e)>0\}$ does not contain rainbows—a feasible rainbow-free solution may not even exist—it eliminates rainbows in subproblems we are interested in, as we will see later. Clearly, if LP (2) is not feasible, we know that the reference problem has no feasible solution.

In all what follows, we only work on edges in $\mathrm{supp}(x^*)$. Therefore, to simplify the exposition, we assume from now on that $E=\mathrm{supp}(x^*)$. This can easily be achieved by deleting all edges $e\in E$ with $x^*(e)=0$ from G.

Our algorithm decomposes the problem of finding an $O(1)$-approximate spanning tree $T\subseteq E$ into an independent family of a special type of spanning tree problem on rainbow-free graphs. To decompose the problem, we consider tight spanning tree constraints. More precisely, let $\mathcal{L}\subseteq 2^V$ be any maximal laminar family of vertex-sets corresponding to spanning tree constraints that are tight with respect to x^*. In other words, $\mathcal{L}$ is maximal laminar family chosen from the sets $L\subseteq V$ satisfying $x^*(E[L])=|L|-1$, where, $E[L]\subseteq E$ denotes the set of edges with both endpoints in L. In particular, $\mathcal{L}$ contains all singletons. We say that $L_2\in\mathcal{L}$ is a child of $L_1\in\mathcal{L}$ if $L_2\subsetneq L_1$ and there is no set $L_3\in\mathcal{L}$ with $L_2\subsetneq L_3\subsetneq L_1$. For $L\in\mathcal{L}$, we denote by $\mathcal{C}(L)\subset\mathcal{L}$ the set of all children of L. Notice that $\mathcal{C}(L)$ forms a partition of L.

To construct a spanning tree T in G we will determine for each $L\in\mathcal{L}$ a set of edges T_L in

$$E_L := E[L]\setminus(\cup_{C\in\mathcal{C}(L)}E[C]),$$

that form a spanning tree in the graph G_L obtained from the graph (L,E_L) by contracting all children of L. Hence, the vertex set of G_L is $\mathcal{C}(L)$, and an original edge $\{u,v\}\in E_L$ is simply interpreted as an edge between the two children $C_u,C_v\in\mathcal{C}(L)$ that contain u and v, respectively. For singletons $L\in\mathcal{L}$, we set $T_L=\emptyset$. One can easily observe that a family $\{T_L\}_{L\in\mathcal{L}}$ of spanning trees in $\{G_L\}_{L\in\mathcal{L}}$ leads to a spanning tree $T=\cup_{L\in\mathcal{L}}T_L$ in G. Constructing "good" spanning trees T_L in G_L, for each $L\in\mathcal{L}$, will be our independent subproblems. As we will argue more formally later, the main benefit of this division is that the edge set E_L used in the subproblem to find T_L does not contain any rainbows. Our goal is to define constraints that the spanning trees T_L have to satisfy, that allow us to conclude that the resulting spanning tree $T=\cup_{L\in\mathcal{L}}T_L$ does not violate the chain constraints by more than a constant factor.

One of the arguably most natural constraint families to impose would be to require that the contribution of T_L to any cut S_i is within a constant factor of the contribution of x^* on S_i when only considering edges in E_L, i.e.,

$$|T_L\cap\delta(S_i)|\leq O(x^*(\delta(S_i)\cap E_L)). \tag{3}$$

If the above inequality holds for each $L\in\mathcal{L}$ and $i\in[n-1]$, then the final spanning tree T will indeed not violate any chain constraint by more than a constant factor: it suffices to sum up the inequalities for a fixed i over all sets L and observe that $\{T_L\}_{L\in\mathcal{L}}$ partitions T, and $\{E_L\}_{L\in\mathcal{L}}$ is

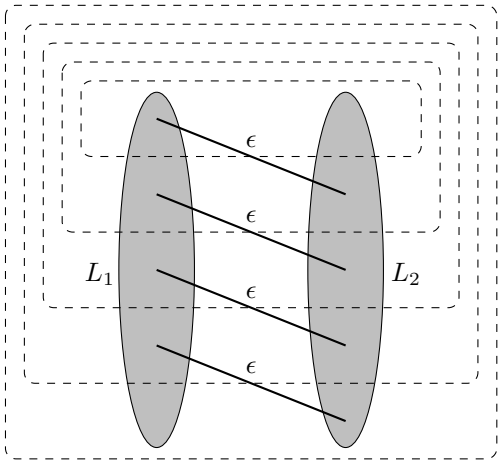

Figure 1: The situation that makes (3) unattainable in general. Shown is a subproblem L with two children L_1 and L_2; for the chain constraints shown as dashed boxes, $x^*(\delta(S_i) \cap E_L) = \epsilon$. This can occur even though there must be at least one unit of weight connecting L_1 and L_2.

a partition of E_L:

$$\begin{aligned} |T \cap \delta(S_i)| &= \sum_{L \in \mathcal{L}} |T_L \cap \delta(S_i)| \leq O\left(\sum_{L \in \mathcal{L}} x^*(\delta(S_i) \cap E_L)\right) \\ &= O(x^*(\delta(S_i))) = O(1)b_i. \end{aligned} \tag{4}$$

Unfortunately, it turns out that it is in general impossible to find spanning trees T_L that satisfy (3). This is because there can be many constraints S_i for which $x^*(\delta(S_i \cap E_L)) = o(1)$, in a setting where one has to include at least one edge in T_L that crosses one of these constraints; see Fig. 1.

We therefore introduce a weaker condition on T_L. For $L \in \mathcal{L}$ and $i \in [n-1]$, let $\mathcal{C}_i(L) \subseteq \mathcal{C}(L)$ be the family of all children $C \in \mathcal{C}(L)$ of L that cross the cut S_i, i.e., $S_i \cap L \neq \emptyset$ and $L \setminus S_i \neq \emptyset$. We want the sets T_L to satisfy the following:

$$|T_L \cap \delta(S_i)| \leq 7 \cdot x^*(\delta(S_i) \cap E_L) + 2 \cdot \mathbf{1}_{\{|\mathcal{C}_i(L)| \geq 2\}} \quad \forall i \in [n-1]. \tag{5}$$

Here, $\mathbf{1}_{\{|\mathcal{C}_i(L)| \geq 2\}}$ is the indicator that is equal to 1 if $|\mathcal{C}_i(L)| \geq 2$ and 0 otherwise.

We first show in Section 2.1 that satisfying the above condition indeed leads to a good spanning tree T.

Theorem 2.1. *For $L \in \mathcal{L}$, let T_L be a spanning tree in G_L that satisfies (5). Then $T = \cup_{L \in \mathcal{L}} T_L$ is a spanning tree in G satisfying*

$$|T \cap \delta(S_i)| \leq 9x^*(\delta(S_i)) \leq 9b_i \quad i \in [n-1].$$

We then show in Section 2.2 that such spanning trees can indeed be found efficiently.

Theorem 2.2. *For each $L \in \mathcal{L}$, we can efficiently find a spanning tree T_L in G_L satisfying (5).*

Combining the above two theorems immediately leads to an efficient algorithm to find a spanning tree in G that violates each chain constraint by at most a factor of 9 whenever LP (2) is feasible, and thus proves Theorem 1.1. For convenience, a summary of our algorithm is provided below.

Algorithm to find spanning tree T that violates chain constraints by a factor of at most 9.

1. Compute an optimal solution x^* to the linear program (2).
2. Independently for each $L \in \mathcal{L}$, invoke Theorem 2.2 to obtain a spanning tree T_L in G_L satisfying (5).
3. Return $T = \cup_{L\in\mathcal{L}} T_L$.

2.1 Analysis of algorithm (proof of Theorem 2.1)

For each $L \in \mathcal{L}$, let T_L be a spanning tree in G_L that satisfies (5), let $T = \cup_{L\in\mathcal{L}} T_L$, and let $i \in [n-1]$. Using the same reasoning as in (4) we can bound the load on chain constraint i as follows:

$$\begin{aligned}|T \cap \delta(S_i)| &= \sum_{L\in\mathcal{L}} |T_L \cap \delta(S_i)| \overset{(5)}{\leq} 7 \sum_{L\in\mathcal{L}} x^*(\delta(S_i) \cap E_L) + 2 \sum_{L\in\mathcal{L}} \mathbf{1}_{\{|\mathcal{C}_i(L)|\geq 2\}} \\ &= 7x^*(\delta(S_i)) + 2 \sum_{L\in\mathcal{L}} \mathbf{1}_{\{|\mathcal{C}_i(L)|\geq 2\}},\end{aligned}$$

using the fact that $\{E_L\}_{L\in\mathcal{L}}$ partitions E. To prove Theorem 2.1, it thus suffices to show

$$\sum_{L\in\mathcal{L}} \mathbf{1}_{\{|\mathcal{C}_i(L)|\geq 2\}} \leq x^*(\delta(S_i)), \tag{6}$$

which then implies

$$|T \cap \delta(S_i)| \leq 9x^*(\delta(S_i)) \leq 9b_i,$$

where the last inequality follows from x^* being feasible for (2). We complete the analysis by showing the following result, which is a stronger version of (6).

Lemma 2.3.

$$\sum_{L\in\mathcal{L}} (|\mathcal{C}_i(L)| - 1)^+ \leq x^*(\delta(S_i)),$$

where $(\cdot)^+ = \max(0, \cdot)$.

Proof. Let $\mathcal{L}_i \subseteq \mathcal{L}$ be the family of all sets in $\mathcal{L}$ that cross S_i, and let $\mathcal{L}_i^{\min} \subseteq \mathcal{L}_i$ be all minimal sets of $\mathcal{L}_i$. We will show that

$$\sum_{L\in\mathcal{L}} (|\mathcal{C}_i(L)| - 1)^+ = |\mathcal{L}_i^{\min}| - 1. \tag{7}$$

Let us first see that this is a strengthening of the lemma. Since all sets $W \in \mathcal{L}_i$ correspond to tight spanning tree constraints with respect to x^*, we have that the restriction $x^*|_{E[W]}$ of x^* to the edges in the graph $G[W]$ is a point in the spanning tree polytope of $G[W]$. In particular, at least one unit of $x^*|_{E[W]}$ crosses any cut in $G[W]$. Since $W \in \mathcal{L}_i$, the set S_i induces a cut $(S_i \cap W, W \setminus S_i)$ in $G[W]$. Hence

$$x^*(\delta(S_i) \cap E[W]) \geq 1 \quad \forall\, W \in \mathcal{L}_i.$$

Now observe that due to minimality of the sets in $\mathcal{L}_i^{\min}$, all sets in $\mathcal{L}_i^{\min}$ are disjoint. Thus

$$x^*(\delta(S_i)) \geq \sum_{W\in\mathcal{L}_i^{\min}} x^*(\delta(S_i) \cap E[W]) \geq |\mathcal{L}_i^{\min}|,$$

which, together with (7), implies Lemma 2.3. Hence, it remains to show (7).

Let $\mathcal{L}_i^{\text{nm}} = \mathcal{L}_i \setminus \mathcal{L}_i^{\min}$ be all sets in $\mathcal{L}_i$ that are not minimal. Notice that only sets $L \in \mathcal{L}^{\text{nm}}$ can have a strictly positive contribution to the left-hand side of (7) since these are precisely the sets $L \in \mathcal{L}$ with $|\mathcal{C}_i(L)| \geq 1$: for any other set $L \in \mathcal{L}$, either (i) $L \notin \mathcal{L}_i$, in which case none of its children can cross S_i since not even L crosses S_i, or (ii) $L \in \mathcal{L}_i^{\min}$, in which case we again get $|\mathcal{C}_i(L)| = 0$ since L has no children in $\mathcal{L}_i$ due to minimality. We thus obtain

$$\sum_{L \in \mathcal{L}} (|\mathcal{C}_i(L)| - 1)^+ = \sum_{L \in \mathcal{L}_i^{\text{nm}}} (|\mathcal{C}_i(L)| - 1). \tag{8}$$

Observe that $\sum_{L \in \mathcal{L}_i^{\text{nm}}} |\mathcal{C}_i(L)|$ counts each set in $\mathcal{L}_i$ precisely once, except for the set $V \in \mathcal{L}_i$ which is the only set in $\mathcal{L}_i$ that is not a child of some other set in $\mathcal{L}_i$. Hence

$$\sum_{L \in \mathcal{L}_i^{\text{nm}}} |\mathcal{C}_i(L)| = |\mathcal{L}_i| - 1. \tag{9}$$

Finally, combining (8) with (9) we obtain

$$\sum_{L \in \mathcal{L}} (|\mathcal{C}_i(L)| - 1)^+ = \sum_{L \in \mathcal{L}_i^{\text{nm}}} (|\mathcal{C}_i(L)| - 1) = |\mathcal{L}_i| - 1 - |\mathcal{L}_i^{\text{nm}}| = |\mathcal{L}_i^{\min}| - 1,$$

thus proving (7). □ □

2.2 Main step of algorithm (proof of Theorem 2.2)

Let $L \in \mathcal{L}$. We now consider the problem of finding a spanning tree T_L in G_L that satisfies (5). Recall that G_L is obtained from the graph (L, E_L) by contracting all children of L. For simplicity, we again interpret an edge $\{v_i, v_j\} \in E_L$ as an edge in G_L between the two vertices corresponding to the sets $C_i, C_j \in \mathcal{L}$ that contain v_i and v_j, respectively.

We start by showing that there are no rainbows in E_L, which is a crucial assumption in the algorithm to be presented in the following.

Lemma 2.4. *For $L \in \mathcal{L}$, E_L does not contain any rainbows.*

Proof. Since $\mathcal{L}$ is a maximal laminar family of tight spanning tree constraints in G with respect to x^*, the spanning tree constraints imposed by the sets in $\mathcal{L}$ define the minimal face of P_{ST} on which x^* lies (this is a well-known result that can be proven through combinatorial uncrossing, see e.g. [**?**]). From this we can conclude that any other tight spanning tree constraint $x^*(E[W]) = |W| - 1$, for some set $W \subseteq V, W \notin \mathcal{L}$, is implied by the spanning tree constraints given by $\mathcal{L}$.

Now assume by the sake of contradiction that there are two edges $e_1, e_2 \in E_L$ that form a rainbow, and let e_1 be the edge contained in strictly more chain constraints than e_2. We will argue that one could slightly increase the fractional value $x^*(e_2)$ by some small value $\delta > 0$ and decrease $x^*(e_1)$ by δ to get a new feasible solution $y^\delta = x^* + \delta \cdot \chi(e_2) - \delta \cdot \chi(e_1)$ to (2) with strictly smaller objective value than x^*, thus violating that x^* is an optimal solution of (2). Clearly, since x^* does not violate any chain constraint, also y does not violate any chain constraint. Furthermore y^δ, for any $\delta > 0$, has indeed a lower objective value than x^*. The only point that remains to show is that there is a small $\delta > 0$ such that $y^\delta \in P_{ST}$. Since all tight spanning tree constraints are implied by the constraints that correspond to sets in $\mathcal{L}$, it suffices to check that y^δ does not violate any of the spanning tree constraints induced by $\mathcal{L}$. This indeed holds since any set $W \in \mathcal{L}$ either satisfies $e_1, e_2 \in E[W]$—this happens if W is a set

containing L—or $e_1, e_2 \notin E[W]$,which happens for any other set $W \in \mathcal{L}$. Hence, for any $W \in \mathcal{L}$ and $\delta > 0$, we have

$$y^\delta(E[W]) = x^*(E[W]) = |W| - 1.$$

Thus, none of the spanning tree constraints that are tight with respect to x^* will by violated by y^δ for any $\delta > 0$. Hence, by choosing a sufficiently small $\delta > 0$ that makes sure that $y^\delta \geq 0$, and that no other (non-tight) spanning tree constraints are violated, we obtain $y^\delta \in P_{ST}$. □ □

We classify chain constraints S_i into two types, depending on the right-hand side of (5). Call a cut S_i *bad* if one can include at most one edge that crosses S_i in T_L without violating (5), i.e.,

$$7x^*(\delta(S_i) \cap E_L) + 2 \cdot \mathbf{1}_{\{|\mathcal{C}_i(L)|\geq 2\}} < 2.$$

Otherwise, call the cut S_i *good*. Notice that for a cut S_i to be bad, we need to have $|\mathcal{C}_i(L)| = 1$ because of the following. Clearly, if $|\mathcal{C}_i(L)| \geq 2$, then S_i cannot be bad due to the term $2 \cdot \mathbf{1}_{\{|\mathcal{C}_i(L)|\geq 2\}}$. If $|\mathcal{C}_i(L)| = 0$, then we use the fact that all edges in $E[L]$ that cross S_i are part of E_L, hence

$$x^*(\delta(S_i) \cap E_L) = x^*(\delta(S_i) \cap E[L]) \geq 1,$$

where the last inequality follows from the fact that $x^*|_{E[L]}$ is in the spanning tree polytope of the graph $(L, E[L])$. Hence a cut S_i is bad if and only if the following two conditions hold simultaneously:

1. $|\mathcal{C}_i(L)| = 1$,
2. $x^*(\delta(S_i) \cap E_L) < \frac{2}{7}$.

An edge $e \in E_L$ is called *bad* if e crosses at least one bad cut S_i, otherwise it is called *good*. We denote by $A_L \subseteq E_L$ the sets of all good edges.

The procedure we use to find a tree T_L satisfying (5) constructs a tree T_L that consists of only good edges, i.e., $T_L \subseteq A_L$. We determine T_L using a matroid intersection problem that asks to find a spanning tree in G_L satisfying an additional partition matroid constraint.

To define the partition matroid we first number the edges $A_L = \{e_1, \ldots, e_k\}$ as follows. For $e \in A_L$, let $\alpha(e) < \beta(e)$ be the lower and higher index of the two endpoints of e, hence, $e = \{v_{\alpha(e)}, v_{\beta(e)}\}$. (Notice that $\alpha(e) = \beta(e)$ is not possible since $x^*(e) > 0 \; \forall e \in E$ and $x^* \in P_{ST}$.) The edges $e \in A_L$ are numbered lexicographically, first by increasing value of $\alpha(e)$ and then by increasing value of $\beta(e)$, i.e., for any $p \in [k-1]$ either $\alpha(e_p) < \alpha(e_{p+1})$, or $\alpha(e_p) = \alpha(e_{p+1})$ and $\beta(e_p) \leq \beta(e_{p+1})$. Note that since A_L has no rainbows, the set of edges in A_L crossing a given S_i are labeled consecutively. Ideally, we would like to group the edges in A_L into consecutive blocks $\{e_p, e_{p+1}, \ldots, e_q\}$ each having a total weight of exactly $x^*(\{e_p, \ldots, e_q\}) = 3/7$. Since this is in general not possible, we will split some of the edges by creating two parallel copies. More precisely, to define the first set P_1 of our partition, let $p \in [k]$ the lagest index for which $x^*(\{e_1, \ldots, e_p\}) \leq 3/7$. If $x^*(\{e_1, \ldots, e_p\}) = 3/7$ then $P_1 = \{e_1, \ldots, e_p\}$. Otherwise, we replace the edge e_{p+1} by two parallel copies e'_{p+1}, e''_{p+1} of e_{p+1}, and we distribute the weight of $x^*(e_{p+1})$ on e'_{p+1}, e''_{p+1} as follows:

$$x^*(e'_{p+1}) = \frac{3}{7} - x^*(\{e_1, \ldots, e_p\}),$$
$$x^*(e''_{p+1}) = x^*(e_{p+1}) - x^*(e'_{p+1}).$$

This splitting operation does not violate any previous assumptions: the weight x^* on the new edge set $\{e_1, \ldots, e_p, e'_{p+1}, e''_{p+1}, e_{p+2}, \ldots, e_k\}$ is still a point in the spanning tree polytope of the graph over the vertices $\mathcal{C}(L)$ with the new edge set. By applying this splitting operation whenever necessary, we can assume that $A_L = \{e_1, \ldots, e_k\}$ can be partitioned into sets $P_1 = \{e_1, \ldots, e_{p_1}\}$, $P_2 = \{e_{p_1+1}, \ldots, e_{p_2}\}$, $\ldots$, $P_s = \{e_{p_{s-1}+1}, \ldots, e_k\}$ satisfying:

(i) $x^*(P_h) = 3/7 \quad \forall h \in [s-1]$,
(ii) $x^*(P_s) \leq 3/7$.

Using this partition we define a unitary partition matroid $M = (A_L, \mathcal{I})$ on the good edges A_L, with independent sets

$$\mathcal{I} = \{U \subseteq A_L \mid |U \cap P_h| \leq 1 \; \forall h \in [s]\}.$$

The tree spanning T_L in G_L that our algorithm selects is any spanning tree $T_L \subseteq A_L$ in G_L that is independent in the partition matroid M. Notice that if there exists a spanning tree in G_L that is independent in M, then such a spanning tree can be found in polynomial time by standard matroid intersection techniques (see [**?**, Volume B] for more details about matroids in general and techniques to find common independent sets in the intersection of two matroids). Hence to complete the description and analysis of our algorithm, all that remains is to show the existence of a spanning tree in G_L that is independent in M, and that such a spanning tree satisfies (5). We address these two statements in the following.

The theorem below shows the feasibility of the matroid intersection problem.

Theorem 2.5. *There exists a spanning tree $T_L \subseteq A_L$ in G_L that is independent in M, i.e., $T_L \in \mathcal{I}$.*

Proof. Let $y \in [0,1]^{A_L}$ be defined by $y(e) = \frac{7}{3}x^*(e)$ for $e \in A_L$. To prove the theorem, we show that y is simultaneously in the matroid polytope of M and in the dominant[2] of the spanning tree polytope of G_L^A, where G_L^A is the graph obtained from G_L by deleting all bad edges. This implies that the intersection of $P_{ST}(G_L^A)$ and the matroid polytope P_M of M is nonempty. Since $P_{ST}(G_L^A) \cap P_M$ is a face of the matroid intersection polytope corresponding to intersecting the matroid M with the graphic matroid on G_L^A, it is therefore integral [**?**]. Hence, if $P_{ST}(G_L^A) \cap P_M$ is nonempty, it contains an integral point, and this corresponds to a spanning tree that is independent in M.

The vector y is clearly in the matroid polytope of the partition matroid M, since for any partition P_h with $h \in [s]$ we have $y(P_h) = \frac{7}{3}x^*(P_h) \leq 1$.

To show that y is in the dominant of the spanning tree polytope of G_L^A, we first discuss some structural properties of G_L^A that allow us to decompose the problem. Let $S_{i_1}, S_{i_2}, \ldots, S_{i_p}$ be all bad cuts, where $1 \leq i_1 < \cdots < i_p \leq n-1$. For $j \in [p]$, let $C_j \in \mathcal{C}(L)$ be the child that crosses S_{i_j}. Notice that since S_{i_j} is bad, there is indeed precisely one child of L that crosses S_{i_j}, and furthermore, there are no good edges crossing S_{i_j}. Hence, every C_j for $j \in [p]$ is a cut vertex in G_L^A, whose removal splits G_L^A into a part where all vertices are contained in S_{i_j} and a part where all vertices are outside of S_{i_j}.

We define the following vertex sets of G_L^A, which correspond to the vertex sets between those cut vertices, including the cut vertices themselves:

$$\begin{aligned}
\mathcal{B}^1 &= \{C \in \mathcal{C}(L) \mid C \cap S_{i_1} \neq \emptyset\}, \\
\mathcal{B}^j &= \{C \in \mathcal{C}(L) \mid C \cap (S_{i_j} \setminus S_{i_{j-1}}) \neq \emptyset\} && \forall j \in \{2, \ldots, p\}, \\
\mathcal{B}^{p+1} &= \{C \in \mathcal{C}(L) \mid C \cap (V \setminus S_{i_p}) \neq \emptyset\}.
\end{aligned}$$

Hence, the above sets contain all vertices of G_L^A precisely once except for the cut vertices, which appear in at least two sets.

For $j \in [p+1]$, let $G_L^j = G_L^A[\mathcal{B}^j]$, i.e., G_L^j is the induced subgraph of G_L^A over the vertices $\mathcal{B}^j$; also let A_L^j denote the edge set of G_L^j (see Fig. 2). To show that y is in the dominant of the spanning tree polytope of G_L^A, we show that the restriction of y to the edges of any of the graphs G_L^j for $j \in [p+1]$ is in the dominant of the spanning tree polytope of G_L^j.

[2]Recall that the dominant of a polyhedron P is the set of vectors x such that $x \geq y$ for some $y \in P$.

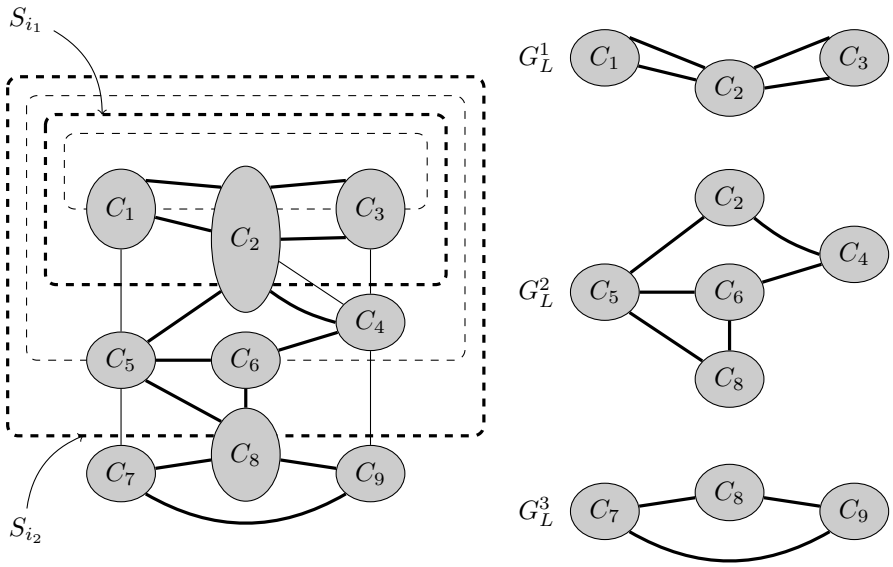

Figure 2: An example showing G_L on the left (with the edges of G_L^A in bold), and the resulting $G_L^j = (\mathcal{B}^j, A_L^j)$ for $j \in \{1, 2, 3\}$.

Fix $j \in [p+1]$. Let $y^j = y|_{A_L^j}$ be the restriction of y to the edges in G_L^j. To show that y^j is in the dominant of the spanning tree polytope of G_L^j, we use the following well-known partition-based description of the dominant of the spanning tree polytope (see [**?**]). Let $\mathcal{W} = \{W_1, \ldots, W_q\}$ be a partition of the sets in $\mathcal{B}^j$, i.e., the vertices of G_L^j, and we denote by $A_L^j(\mathcal{W})$ all edges of A_L^j with endpoints in two different sets of the partition $\mathcal{W}$. To show that y^j is in the dominant of the spanning tree polytope of G_L^j, we have to show that the following inequality holds for any partition $\mathcal{W} = \{W_1, \ldots, W_q\}$ of $\mathcal{B}^j$:

$$y(A_L^j(\mathcal{W})) \geq q - 1. \tag{10}$$

Given a partition $\mathcal{W} = \{W_1, \ldots, W_q\}$ of $\mathcal{B}^j$ with $q \geq 2$, we define a partition $\mathcal{Z} = \{Z_1, \ldots, Z_q\}$ of the set $\mathcal{C}(L)$ as follows. We start by setting $Z_r = W_r$ for $r \in [q]$. If $j \in \{2, \ldots, p+1\}$, we add $\cup_{s=1}^{j-1} \mathcal{B}^s$ to the set $Z \in \mathcal{Z}$ that contains C_{j-1}. Additionally, if $j \in [p]$, we add $\cup_{s=j+1}^{p+1} \mathcal{B}^s$ to the set $Z \in \mathcal{Z}$ that contains C_j. Hence, $\mathcal{Z}$ is identical to $\mathcal{W}$ with the possible exception of up to two sets. Let $E_L(\mathcal{Z})$ be the set of all edges in E_L that cross the partition $\mathcal{Z}$. Notice that $E_L(\mathcal{Z})$ consists of all edges in $A_L^j(\mathcal{W})$ together with all bad edges that cross either of the bad cuts $S_{i_{j-1}}$ or S_{i_j}. Since the x^*-weight of the set of all edges crossing any bad cut is bounded by $\frac{2}{7}$ we obtain

$$x^*(E_L(\mathcal{Z})) \leq x^*(A_L^j(\mathcal{W})) + 2 \cdot \frac{2}{7}. \tag{11}$$

Furthermore, since x^* is in the spanning tree polytope of G_L, it fulfills the partition-constraints that define the dominant of the spanning tree polytope of G_L. For the partition $\mathcal{Z}$ this leads to

$$x^*(E_L(\mathcal{Z})) \geq q - 1. \tag{12}$$

Using the definition of y and combining (11) and (12) we obtain

$$y(A_L^j(\mathcal{W})) = \frac{7}{3} x^*(A_L^j(\mathcal{W})) \overset{(11)}{\geq} \frac{7}{3} x^*(E_L(\mathcal{Z})) - \frac{4}{3} \overset{(12)}{\geq} \frac{7}{3}(q-1) - \frac{4}{3} \geq q - 1,$$

since $q \geq 2$. This shows (10) and therefore completes the proof. □ □

The following theorem finishes the analysis of our algorithm.

Theorem 2.6. *Let $T_L \subseteq A_L$ be a spanning tree in G_L that is independent in M. Then T_L satisfies (5).*

Proof. Consider a cut S_i for some fixed $i \in [n-1]$. We consider the partition $P_1, \ldots, P_s$ of A_L used to define the partition matroid M. We are interested in all sets in this partition that contain edges crossing S_i. Recall that the edges crossing S_i are consecutively labelled. Thus the sets of the partition containing edges crossing S_i are also consecutively numbered, so let these be $P_a, P_{a+1}, \ldots, P_b$, where $1 \leq a \leq b \leq s$. Since T_L contains at most one edge in each partition, we have

$$|T_L \cap \delta(S_i)| \leq b - a + 1. \tag{13}$$

We first consider the case $b - a \geq 2$. Notice that all edges in any set P_h for $a < h < b$ cross S_i. Hence,

$$x^*(\delta(S_i) \cap E_L) \geq \sum_{h=a+1}^{b-1} x^*(P_h) = (b - a - 1) \cdot \frac{3}{7},$$

where we used $x^*(P_h) = \frac{3}{7}$ for $1 \leq h \leq s-1$. Combining the above inequality with (13), and using that $b - a \geq 2$ in the second inequality, we obtain that

$$|T_L \cap \delta(S_i)| \leq b - a + 1 \leq 3(b - a - 1) \leq 7x^*(\delta(S_i) \cap E_L).$$

Thus T_L satisfies (5).

Assume now $b - a \leq 1$. If S_i is bad, then $|T_L \cap \delta(S_i)| = 0$ since T_L only contains good edges and no good edge crosses any bad cut. Hence, T_L trivially satisfies (5). So assume that S_i is good, i.e., either $|\mathcal{C}_i(L)| \geq 2$ or $x^*(\delta(S_i) \cap E_L) \geq \frac{2}{7}$. If $|\mathcal{C}_i(L)| \geq 2$, then beginning again from (13) we have

$$|T_L \cap \delta(S_i)| \leq b - a + 1 \leq 2 = 2 \cdot \mathbf{1}_{|\mathcal{C}_i(L)| \geq 2}.$$

Otherwise, if $x^*(\delta(S_i) \cap E_L) \geq \frac{2}{7}$, then

$$|T_L \cap \delta(S_i)| \leq 2 \leq 7x^*(\delta(S_i) \cap E_L).$$

Either way, T_L satisfies (5). □ □

3 Hardness and integrality gaps

In this section, we provide the proof of Theorem 1.2.

3.1 The chain-constrained partition problem

We will begin by considering a different problem, where we replace the spanning tree constraint by a unitary partition matroid. We will show integrality gaps and hardness for this problem first, and then show how this can be leveraged to the spanning tree setting via a gadget.

So consider the following problem, that we call the *chain-constrained partition problem*. We are given a graph $G = (V, E)$ and a chain $S_1 \subsetneq S_2 \subsetneq \cdots \subsetneq S_\ell$ of cuts, with associated upper bounds b_i for $i \in [\ell]$. We are also given a partition $\{B_1, B_2, \ldots, B_q\}$ of E. The goal is to pick precisely one edge from each part, while satisfying the chain constraints. Without loss

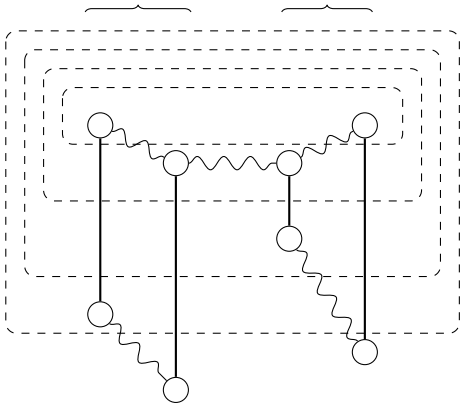

Figure 3: First step of the reduction to the chain-constrained partition problem. Edges E' are shown as wavy; the dashed sets are the chain constraints.

of generality, we can take E to be a matching (by splitting vertices as needed); we will always assume this in what follows.

The problem has a completely equivalent formulation as follows. We are given a $\ell \times m$ consecutive-ones matrix A (i.e., entries of A are 0-1, and in any column of A, the 1's are all consecutive) and an integral right-hand side vector b of length ℓ. We are also given a partition $\mathcal{B} = \{B_1, \ldots, B_q\}$ of $[m]$. The goal is to pick one column from each part so that the sum of the chosen columns does not exceed b, if such a choice exists; or in other words, to find a vector $x \in \{0,1\}^m$ such that $Ax \leq b$ and for $i \in [q]$ there is precisely one index $j \in B_i$ with $x_j = 1$. The correspondence to the other formulation is that each row corresponds to a chain constraint, and each column of 1's to an edge.

Relation to the chain-constrained spanning tree problem. Here we will show that additive integrality gaps and hardness results for the chain-constrained partition problem transfer to the chain-constrained spanning tree problem. Note that only *additive* results will transfer; our gadgets will require increasing the right hand side, and constant-factor multiplicative hardness (which is an easy consequence for the partition version) will not carry over.

So let an instance of the chain-constrained partition problem be given, with $G = (V, E)$ being the graph. For each edge e, let $\alpha(e)$ be the innermost endpoint of edge e (with respect to the chain) and $\beta(e)$ the outermost endpoint. Now construct the set of edges E' as the union of an arbitrary spanning tree on $\{\alpha(e) \mid e \in E\}$, and for each $i \in [q]$, an arbitrary spanning tree on $\{\beta(e) \mid e \in B_i\}$ (see Fig. 3). Keep the same set of chain constraints, but modify the upper bounds by setting $b'_i = b_i + |\delta(S_i) \cap E'|$.

Now consider the chain-constrained spanning tree problem on $G' = (V, E \cup E')$, but subject to the extra restriction that *every edge in E' must be picked*. This is clearly precisely the same problem as the original chain-constrained partition problem: $R \subseteq E$ satisfies the partition constraints if and only if $R \cup E'$ is a spanning tree of G', and the definition of b'_i absorbs the change in the number of edges across the cut $\delta(S_i)$. In order to eliminate this extra restriction, we need one further trick.

Let $s = |E'| + 1$. For each edge $e = \{v_a, v_b\} \in E'$, with $a < b$, we split e into a path of length $s\ell$. Let $v_e^{(r)}$ denote the vertices of this path in order, with $v_e^{(0)} = v_a$ and $v_e^{(s\ell)} = v_b$. Call the resulting graph G'', and let E'' be the set of edges which replaced E'. Note that

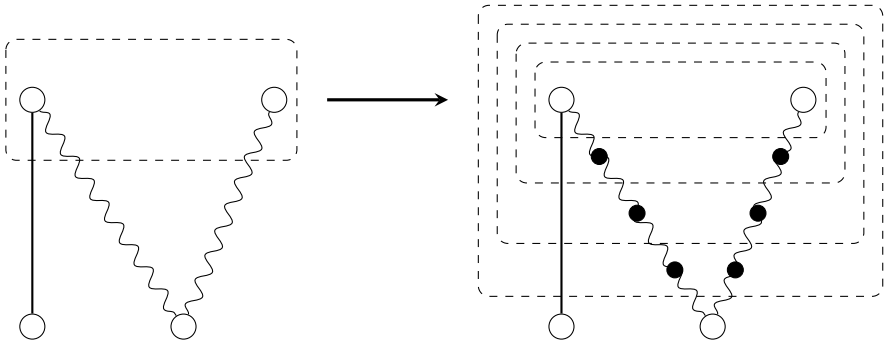

Figure 4: Edge splitting ensures that there is no advantage to not taking all edges in E''.

(1) for each edge in E', all but at most one of the edges in the corresponding path must be chosen in any spanning tree of G''.

We now define, for $i \in [\ell]$ and $j \in [s]$,

$$S_i^{(j)} = S_i \cup \left\{v_e^{(k)} : e \in E'[S_i], k \in [s\ell]\right\} \cup \left\{v_e^{(k)} : e \in \delta_{G'}(S_i), k < is + j\right\}.$$

(See Fig. 4.) This family of sets clearly does form a chain. We may thus define an instance of the chain-constrained spanning tree problem on the graph G'', and with the contraints $|T \cap \delta(S_i^{(j)})| \leq b'_i$ for all $i \in [\ell]$, $j \in [s]$. Call this problem the derived problem.

Lemma 3.1. *If the original chain-constrained partition problem has a feasible solution, then so does the derived chain-constrained spanning tree problem. Furthermore if T is an additive k-approximate solution to the derived problem, then $T \cap E$ is an additive k-approximate solution to the original problem.*

Proof. First, recall that if R is a feasible solution to the original problem in G, then $R \cup E'$ is a spanning tree in G'; hence $T'' := R \cup E''$ is a spanning tree in G''. Moreover

$$|T'' \cap \delta_{G''}(S_i^{(j)})| = |R \cap \delta_G(S_i)| + |\delta_G(S_i)| \leq b'_i, \text{ for each } i \in [\ell], j \in [s].$$

So T'' is a feasible solution to the derived problem.

Now consider an arbitrary spanning tree T in G''. Focus on some particular $i \in [\ell]$. We claim that

$$|T \cap E \cap \delta_G(S_i)| + |\delta_G(S_i) \cap E'| = \max_{j \in [s]} |T \cap \delta_{G''}(S_i^{(j)})|.$$

This is a consequence of point (1) above; there are at most $|E'| < s$ edges of e' crossing $\delta(S_i)$, and so there must be some choice of $j \in [s]$ for which $T \supseteq E'' \cap \delta_{G''}(S_i^{(j)})$. It follows by the definition of b'_i that if T is an additive k-approximation to the derived problem, then $T \cap E$ is an additive k-approximation to the original problem. □

Note that if the original instance has n vertices, then the derived instance will have $O(ns) = O(n^2)$ vertices. So to prove Theorem 1.2 it suffices to show an $\Omega(\log n / \log\log n)$ additive inapproximability result for the chain-constrained partition problem.

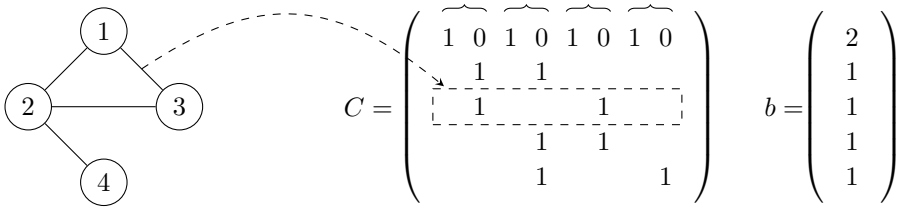

Figure 5: Reduction from independent set to a column selection problem; here $k = 2$.

3.2 NP-hardness

We will now consider only the chain-constrained partition problem from this point, and we will use the equivalent consecutive-ones formulation discussed in Section 3.1. Our first modest goal will be to show that it is NP-complete to decide whether there is a solution that does not violate any constraints at all.

We first note that the same problem, but *without* the requirement that the ones in any column of A must be consecutive, is clearly hard, even if all parts of the partition have size 2. We reduce from the independent set problem (see Fig. 5). Let $H = (W, E)$ be a given graph, and k a given integer, and consider the questiof whether H has an independent set of size k. Let $t = |W|$, and $w_1, \ldots, w_t$ a labelling of W. Define the partition $\{\{2j-1, 2j\} \mid j \in [t]\}$ of W; if column $2j$ is picked, this will represent that vertex j is chosen in the independent set. We define C and b as follows. The first row of C contains a 1 in all columns of odd index, and $b_1 = t - k$; this ensures that at least k vertices are chosen. Each edge $\{w_i, w_j\} \in E$ has one corresponding row in C, consisting of a 1 in columns $2i$ and $2j$, with all other entries 0; the corresponding entry in b is also 1. This represents the constraint that at most one of w_i and w_j can be picked. It is clear that a feasible solution x exists if and only if an independent set of size k exists.

So our goal will be to simulate this pairwise column-selection problem without the consecutive-ones restriction, via an instance where the consecutive-ones property holds. So let C be an arbitrary $\ell \times m$ 0-1 matrix (we may assume each column contains at least one nonzero entry), and b the vector of upper bounds. Assume that the columns are ordered so that column 1 is paired with column 2, column 3 is paired with column 4, and so on. Let r_i be the number of ones in column $i \in [m]$, and let $r' = \sum_{j=1}^{m} r_j$. First expand out the matrix C horizontally, by replacing column j by r_j columns, moving the i'th 1 in the column to the i'th replacement column (in the same row). Call the resulting matrix $\tilde{C}$; so each column of $\tilde{C}$ has a single 1, and the first r_1 columns of $\tilde{C}$ correspond to the first column of C, the following r_2 columns to the second column of C, etc. Let Λ_n denote the $n \times 1$ matrix consisting of all ones, and I_n the

$n \times n$ identity matrix. Let Q be the $r' \times \ell$ matrix

$$Q = \begin{pmatrix} \Lambda_{r_1} & & & \\ & \Lambda_{r_2} & & \\ & & \ddots & \\ & & & \Lambda_{r_k} \end{pmatrix}.$$

Then define the final matrix (which has the consecutive-ones property) by

$$A = \begin{pmatrix} \tilde{C} & & \\ & I_{r'} & Q \end{pmatrix},$$

as well as the partition

$$\big\{\{a, r' + a\} \mid a \in [r']\big\} \cup \big\{\{2r' + 2j - 1, 2r' + 2j\} : j \in [m/2]\big\}.$$

The packing constraint for the first ℓ rows remains unchanged, i.e., is given by b. The packing constraint for all other rows is chosen to be 1.

Now suppose a feasible solution picks column $2r' + 1$. Then by the definition of A, and our choice of packing constraints, none of the columns $r' + 1, r' + 2, \ldots, r' + r_1$ are chosen. Hence *all* of the columns $1, 2, \ldots, r_1$ are chosen. This exactly corresponds to picking column 1 of C. If we do not pick column $2r' + 1$ on the other hand, there is no good reason to choose any of these columns. The same argument shows that if column $2r' + j$ is picked for some $j \in [m]$, then all columns corresponding to column j of C are picked. Since we must pick one of the columns $2r' + 2j - 1$ and $2r' + 2j$, this precisely mimics the requirement that we pick one out of columns $2j - 1$ and $2j$ of C.

Once again, the blowup is only polynomial; if C is $\ell \times m$, then A will be $O(m\ell) \times O(m\ell)$. This completes the proof that the chain-constrained partition problem is NP-complete.

3.3 Boosting to an additive $\Omega(\log n / \log \log n)$ hardness

First, consider the following simple integrality gap construction, which will motivate the hardness construction. Fix an integer k. We will construct a sequence of matrices $A^1, A^2, \ldots$, with A^i having k^i rows and $\Theta(k^i)$ columns, as well as a partition $\mathcal{B}^i$ of the columns of A^i, as follows. Let A^1 be a $k \times k$ identity matrix, and $\mathcal{B}^1$ be the trivial partition with only one part. We construct A^{i+1} inductively from A^i as demonstrated by the block matrix diagram of Fig. 6; the left k columns consist of a vertically stretched $k \times k$ identity matrix (i.e., the Kronecker product $A^1 \otimes \Lambda_k$, where recall Λ_k is a column vector of k ones), and the remaining columns consist of

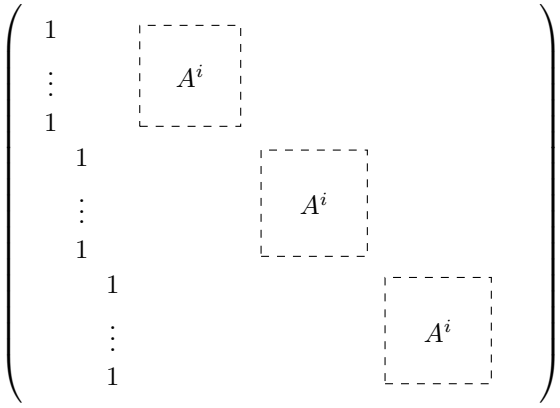

Figure 6: Construction of the matrix A^{i+1} (here, $k = 3$).

the Kronecker product $I_k \otimes A^i$. The partition $\mathcal{B}^{i+1}$ contains one part consisting of the first k columns, and then a copy of $\mathcal{B}^i$ for each set of columns corresponding to a copy of A^i.

Consider now the instance defined by A^k and $\mathcal{B}^k$ with a right hand side consisting of all ones. Observe that the obvious uniform fractional solution, where we pick $x_i = 1/k$ for each i, is feasible, since there are precisely k 1's in each row. However, for any 0-1 solution x satisfying the partition constraints, $\|A^k x\|_\infty \geq k$. Indeed, it is easy to see inductively that $\|A^i x\|_\infty \geq i$ for any 0-1 vector x satisfying the partition constraints $\mathcal{B}^i$.

It is easily checked that $k = \Theta(\log m / \log\log m)$, where m is the number of columns of A^k.

In order to obtain a hardness result, we will now embed the hardness construction from the previous section as a gadget within this integrality gap construction. So let an arbitrary chain-constrained partition problem be given; $A^1 x \leq b^1$, with partition $\mathcal{B}^1$. Let m be the number of columns of A^1, and k the number of rows. We may assume that $k \leq 2m$, since otherwise, due to the consecutive-ones structure, some constraints will necessarily be redundant. We will construct a larger instance such that finding an integral solution with additive violation $o(\log m / \log\log m)$ would provide an exact solution to the starting problem.

The approach is essentially the same as for the integrality gap construction. We inductively define A^{i+1} (with k^{i+1} rows) from A^i (with k^i rows) by

$$A^{i+1} = \begin{pmatrix} A^1 \otimes \Lambda_{k^i} & I_k \otimes A^i \end{pmatrix}. \tag{14}$$

The vector b^{i+1} (of length k^{i+1}) is obtained by taking k copies of b^i (which has length k^i), added to a stretched version of b^1:

$$b^{i+1} = b^1 \otimes \Lambda_{k^i} + \Lambda_k \otimes b^i. \tag{15}$$

Finally, define x^{i+1} so that the first m components of x^{i+1} is a copy of x^1, and the remaining components yield the vector $\Lambda_k \otimes x^i$ (i.e., just x^i repeated k times). This ensures that if $A^i x^i \leq b^i$ and $A^1 x^1 \leq b^1$, then $A^{i+1} x^{i+1} \leq b^{i+1}$. The partition $\mathcal{B}^{i+1}$ is defined in the obvious way: the first m columns are partitioned using $\mathcal{B}^1$, the remaining columns are partitioned using k consecutive partitions of type $\mathcal{B}^i$.

Lemma 3.2. *If we have a* 0-1 *solution* z^i *satisfying* $\mathcal{B}^i$ *and the relaxed constraints*

$$A^i z^i \leq b^i + (i-1) \cdot \Lambda_{k^i}, \tag{16}$$

then we can efficiently construct a 0-1 *solution* z *satisfying* $\mathcal{B}^1$ *and where* $A^1 z \leq b^1$.

Proof. We proceed by induction. The claim clearly holds for $i = 1$.

So assume the claim holds for $i-1$. Let $\tilde{z}$ be the vector consisting of the first m components of z^i. By the definition of $\mathcal{B}^i$, $\tilde{z}$ satisfies $\mathcal{B}^1$. If $A^1\tilde{z} \leq b^1$, we already have the required solution, so suppose not.

Choose t so that $\beta := (A^1\tilde{z})_t \geq b^1_t + 1$. Let m_i denote the number of columns of A_i. Now let z^{i-1} be the vector of length k^{i-1} consisting of the components of z^i from index $m+(t-1)\cdot m_{i-1}$ through $m + t \cdot m_{i-1} - 1$. Fix any $j \in [k^{i-1}]$, and consider row $j' := j + (t-1)\cdot k^{i-1}$ of (16). From (14) we have $(A^i z^i)_{j'} = \beta + (A^{i-1}z^{i-1})_j$, and so

$$(A^{i-1}z^{i-1})_j \leq b^i_{j'} - \beta + (i-1) \overset{(15)}{=} b^{i-1}_j + b^1_t - \beta + (i-1) \leq b^{i-1}_j + (i-2).$$

So z^{i-1} satisfies the conditions of the lemma for $i-1$, and so by induction the required z can be found efficiently. □ □

Notice that A^i has $k^i \leq (2m)^i = O(m)^i$ rows. Furthermore, the number of columns m_i of A^i is equal to $m + k \cdot m_{i-1}$. Since $m_1 = m$, we obtain

$$m_i = m\sum_{j=0}^{i-1} k^j = m\frac{k^i - 1}{k-1} = O(m)^i.$$

Choosing $i = m$, we obtain a matrix A^m with $O(m)^m$ columns and $O(m)^m$ rows. This we can reduce to a chain-constrained spanning tree problem on $m^{O(m)}$ nodes; writing m in terms of the number of nodes n, we find that $m = \Theta(\log n / \log\log n)$, and we obtain the required additive hardness of Theorem 1.2.

4 Conclusions

We would like to close with several interesting directions for future research. One very natural question is whether there is an $O(1)$-approximation for laminar cut constraints; we believe this to be true. Although it seems non-trivial to directly generalize our procedure for the chain-constrained case to the laminar case, we hope that they can be useful in combination with insights from $O(1)$-approximations for the degree-bounded case.

Another natural extension would be to a cost version of the problem, where edges are weighted and the goal is to return a spanning tree of minimum cost satisfying the chain constraints. The main reason our approach does not generalize easily to this setting is that we use a particular objective function to eliminate rainbows in the subproblems. Recently, Linhares and Swamy [**?**] have shown how to produce a spanning tree that violates all chain constraints by a constant multiplicative factor, and in addition has cost within a constant factor of the optimum (the optimum being the cost of a cheapest spanning tree that does not violate any of the constraints at all). Their algorithm relies explicitly on the results of this paper, in particular Theorem 2.1 and Theorem 2.2. It remains an open question to efficiently find a spanning tree that violates the chain constraints by a constant multiplicative factor and has cost no larger than the optimum.

Acknowledgements. We are grateful to the anonymous referee for an extremely careful reading and many constructive comments.