The Algorithmic Power of the Greene-Kleitman Theorem

Shimon Kogan
Weizmann Institute of Science, Rehovot, Israel

Merav Parter
Weizmann Institute of Science, Rehovot, Israel

Abstract

For a given n-vertex DAG $G = (V, E)$ with transitive-closure $TC(G)$, a *chain* is a directed path in $TC(G)$ and an *antichain* is an independent set in $TC(G)$. The *maximum k-antichain* problem asks for computing the maximum k-colorable subgraph of the transitive closure. The related *maximum h-chains* problem asks for computing h disjoint chains (i.e., cliques in $TC(G)$) of largest total lengths. The celebrated Greene-Kleitman (GK) theorem [J. of Comb. Theory, 1976] demonstrates the (combinatorial) connections between these two problems.

In this work we translate the combinatorial properties implied by the GK theorem into *time-efficient* covering algorithms. In contrast to prior results, our algorithms are applied directly on G, and do not require the precomputation of its transitive closure. Let $\alpha_k(G)$ be the maximum number of vertices that can be covered by k antichains. We show:

- For every n-vertex m-edge DAG $G = (V, E)$, one can compute at most $(2k-1)$ disjoint antichains that cover $\alpha_k(G)$ vertices in time $m^{1+o(1)}$ (hence, independent in k). This extends the recent $m^{1+o(1)}$-time Maximum-Antichain algorithm (where $k = 1$) by [Cáceres et al., SODA 2022] to any value of k.
- For every n-vertex m-edge Partially-Ordered-Set (poset) $P = (V, E)$, one can compute $(1+\epsilon)k$ disjoint antichains that cover $\alpha_k(P)$ vertices in time $O(\sqrt{m} \cdot \alpha_k(P) \cdot n^{o(1)}/\epsilon)$, hence at most $n^{2+o(1)}/\epsilon$. This improves over the exact solution of $O(n^3)$ time of [Gavril, Networks 1987] at the cost of producing $(1+\epsilon)k$ antichains instead of exactly k.

The heart of our approach is a linear-time greedy-like algorithm that translates suitable chain collections $\mathcal{C}$ into an *parallel* set of antichains $\mathcal{A}$, in which $|C_j \cap A_i| = 1$ for every $C_j \in \mathcal{C}$ and $A_i \in \mathcal{A}$. The correctness of this approach is underlined by the GK theorem.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Chains, Antichains, DAG

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.80

Funding This project is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 949083).

1 Introduction

For a given n-vertex DAG $G = (V, E)$ with transitive-closure $TC(G)$, a *chain* is a directed path in $TC(G)$ and an *antichain* is an independent set in $TC(G)$. This paper is concerned with time-efficient algorithms for computing a small collection of disjoint antichains that maximizes the number of covered vertices. For an integer k, the maximum number of vertices that can be covered by k-chains (resp., antichains) is denoted hereafter by $\beta_k(G)$ (resp., $\alpha_k(G)$). The relations between chains and antichains have been studied thoroughly over the years, mostly from a combinatorial (or existential) perspective, and more recently from an algorithmic perspective. Most attention has been given for the following covering notions. For a vertex-disjoint subsets of vertices $\mathcal{S} = \{S_1, \ldots, S_\ell\}$, let $V(\mathcal{S}) = \cup_{i=1}^{\ell} S_i$.


© Shimon Kogan and Merav Parter;
licensed under Creative Commons License CC-BY 4.0

- **Maximum Antichain (MA)**: An antichain A of largest size, i.e., such that $|A| = \alpha_1(G)$.
- **Minimum Chain Cover (MCC)**: A set $\mathcal{C}$ with minimal number of chains such that $V(\mathcal{C}) = V(G)$.
- **Max h-Chains** (h-**MC**): Set of h chains $\mathcal{C}$ with $|V(\mathcal{C})| = \beta_h(G)$.
- **Max k-Antichains** (k-**MA**): Set of k antichains $\mathcal{A}$ with $|V(\mathcal{A})| = \alpha_k(G)$.

In a seminal paper, Dilworth [6] draw the first connection between these covering problems for the case of $k = 1$. The celebrated Dilworth theorem states that the size of the maximum antichain equals to the smallest number of chains, ω, into which the vertex-set may be partitioned. This value, ω, is also known as the width of the graph which recently has been shown to play an important role in the computation of reachability shortcuts[1] [12, 11]. Algorithmically, almost-linear time algorithms for the MA and Minimum Chain Cover problems have been provided very recently by Cáceres [2]. The solutions to these two problems are based on a reduction to the Min-Cost Max-Flow problem. It is, however, intriguing to ask:

▶ **Question 1.1.** *Can we use Dilworth theorem to show a direct* time-efficient *algorithmic reduction from maximum antichain to minimum chain covers (or vice verse)?*

We are in particular interested in linear time deterministic reductions. The celebrated Greene-Kleitman (GK) theorem [9] can be viewed as a generalization of Dilworth's theorem for any value of k, by drawing a connection between chain and antichain covers. The theorem is based on the notion of the *k-norm* of a chain collection $\mathcal{C}$, denoted by $||\mathcal{C}||_k$, defined as:

$$||\mathcal{C}||_k = \sum_{C_i \in \mathcal{C}} \min\{|C_i|, k\} \ . \tag{1.1}$$

For $k = 1$, $||\mathcal{C}||_1 = |\mathcal{C}|$. Intuitively, each chain of length at least k in $\mathcal{C}$ contributes $+k$ to the k-norm, and shorter chains C_i contribute $+|C_i|$. The Greene-Kleitman (GK) Theorem can be stated as follows:

▶ **Theorem 1.2** (Greene-Kleitman Theorem, [9])**.** *Let $\mathcal{P} = \{\mathcal{C}' \mid V(\mathcal{C}') = V\}$ be the collection of all possible chain covers for G, then* $\min_{\mathcal{C}' \in \mathcal{P}} ||\mathcal{C}'||_k = \alpha_k(G)$.

That is, the number of vertices covered by maximum k-antichains equals to the minimum k-norm over all chain covers. The GK theorem has been studied thoroughly over the years, and the literature by now exhibits multiple analogous formulations of the theorem. West [20], for example, describes the GK theorem as an equality between pairs of dual integer packing and covering in partially ordered sets[2] (henceforth, *posets*). The chain cover that admits the minimum k-norm is referred to, in many of the prior works, by *k-saturated partition*. Also note that the h-MC problem is equivalent[3] to computing a set of h cliques (in the transitive closure) that covers the maximum number of vertices. Similarly, k-MA corresponds to maximum k-colorable subgraph in the transitive closure. These covering and coloring perspectives are very common in prior work, see e.g., [8].

While the original proof by Greene and Kleitman is based on lattice theory, alternative proofs and genelizations have been provided over the years [10, 13, 1, 4, 18]. Saks [17] presented an elegant combinatorial proof by applying the Dilworth's theorem to the product of a poset with a chain of length k. Algorithmically, this provides a reduction from the k-MA problem to solving the (simpler) MA problem in a graph with kn vertices and km edges.

---

[1] A d-reachability shortcut for a digraph G is a set of edges from $TC(G)$ whose addition to G reduces the directed diameter of G to at most d.
[2] A poset is a special type of a DAG G for which $TC(G) = G$.
[3] As a chain C in $TC(G)$ induces a clique on $V(C)$ in $TC(G)$.

Frank [7] and Gavril [8] further investigated the algorithmic and combinatorial connections between chain and antichain covers by noting that the minimum k-norm cover is tightly connected to the h-MC problem. A recent important development by Cáceres [2] and Kogan and Parter [11] yields an almost linear time algorithm for computing the h-MC. The algorithm h-MC is obtained by a reduction to the Min-Cost Max-Flow problem. While maximum chain covering problems (namely, MCC and h-MC) are currently (almost) settled algorithmically, no subqubic time algorithm is currently known for the k-MA problem. As all current k-MA algorithms are based on the precomputation of the transitive closure, they require $O(n^\omega)$ time, which is the time to compute an $n \times n$ matrix multiplication. The computation of the transitive closure may sound, at first glance, inevitable, as the k-MA problem asks for detecting a maximum k-colored subgraph of the *transitive closure*. Inspired by the recent advances for the dual chain covering problem, we ask whether one can solve the k-MA problem in almost linear time, independent in the number of antichains k. To approach this goal using the GK theorem, we more concretely ask:

▶ **Question 1.3.** *Can we use the Greene-Kleitman theorem to show a direct* time-efficient *algorithmic reduction from maximum k-antichains to maximum h-chains? I.e., can we translate an algorithm for computing a maximum covering by h cliques into an algorithm that computes a maximum k-colorable subgraph in the transitive closure,* without *computing the transitive closure itself?*

We answer Questions 1.1 and 1.3 in the affirmative, up to a small slack (where we output $2k$ antichains rather k), by showing linear time algorithms that translate a given chain collection with certain covering guarantees into a suitable covering collection of antichains.

Our Contributions. Throughout, for a given DAG $G = (V, E)$, let $n = |V|$ and $m = |E|$. The running time of some of our key algorithms is dominated by the time complexity of the min-cost max-flow (MCMF) problem, denoted by $T_{\mathsf{MCMF}}(m)$. Formally, $T_{\mathsf{MCMF}}(m)$ is time to solve a MCMF instance on m-edge directed graph with polynomial costs and capacities. By a recent breakthrough result of [5, 19], $T_{\mathsf{MCMF}}(m) = m^{1+o(1)}$.

By taking an algorithmic Dilworth approach, we translate a given minimum chain cover into a maximum antichain, in *linear* time.

▶ **Theorem 1.4 (Algorithmic Dilworth Theorem).** *There is an algorithm* MaximumAntichain *that given a DAG $G = (V, E)$ and a minimum chain cover $\mathcal{C} = \{C_1, \ldots, C_t\}$ computes a maximum antichain (MA) in $O(|E|)$ time. This in particular implies an MA algorithm in time $\widetilde{O}(T_{\mathsf{MCMF}}(m))$ (hence, almost linear).*

While there is already an almost-linear time algorithm for computing the MA (by [3]), the benefit of Theorem 1.4 is in providing a *linear* reduction between the two problems. This serves also a warm-up for computing the approximate k-MA solution, by exploiting the Greene-Kleitman theorem. Our key result outputs $2k$ antichains that cover $\alpha_k(G)$ vertices in almost-linear time. Or, alternatively, our algorithm computes in almost-linear time k antichains covering at least $\alpha_k(G)/2$ of the vertices. These notions of approximations of the maximum k-colorable subgraphs have been studied in the past, for general graphs [16, 14].

▶ **Theorem 1.5 (Algorithmic Greene-Kleitman Theorem).** *There is an algorithm* ApproxMA *that given a DAG $G = (V, E)$ and an integer parameter k computes a set of $(2k-1)$ vertex-disjoint antichains $\mathcal{A} = \{A_1, \ldots, A_{2k}\}$ with $|V(\mathcal{A})| = \alpha_k(G)$ in time $\widetilde{O}(T_{\mathsf{MCMF}}(m))$. The output of algorithm* ApproxMA *can also provide k disjoint antichains $\mathcal{A}'$ such that $|V(\mathcal{A}')| \geq \alpha_k(G)/2$.*

Figure 1 The road-map from maximum h-chains to approximate maximum k-antichains.

The time complexity should be compared with the state-of-the-art $O(n^3)$-time algorithm of Gavril [8]. Fig. 1 summarizes the key algorithmic steps, which are tightly dictated to the structure provided by the GK theorem.

In Section 4, we combine the algorithm of Saks [17] with the recent results by [3] (and that of Thm. 1.4) to provide a k-MA algorithm that runs in $O(k \cdot m^{1+o(1)})$ time. The main benefit in our $2k$-MA solution of Thm. 1.5 is in providing an almost-linear time algorithm that is *independent* in k. Up to the extra factor of 2, this now matches the time complexity of maximum h-chains and maximum k-antichains. The former has been resolved only recently by the work of Cáceres [2].

A New Ordered Version of GK Theorem. Our greedy-based approach for computing antichains from a given chain collection is based on an ordered variant of the GK theorem, which might find further implications. A collection of vertex-subsets $\bar{\mathcal{S}} = \{S_1, \ldots, S_\ell\}$ is topologically ordered if for every $i < j$, it holds that all the vertices in S_i precede each of the vertices in S_j, in any topological ordering of G. Sets of chains $\mathcal{C}$ and antichains $\mathcal{A}$ are said to be *parallel* if $|C_i \cap A_j| = 1$ for every $C_i \in \mathcal{C}$ and $A_j \in \mathcal{A}$. Note that we use here the term parallel rather than orthogonal, since in the literature the term orthogonal has the extra requirement that $V(\mathcal{C}) \cup V(\mathcal{A}) = V$.

▶ **Lemma 1.6** (Ordered Greene-Kleitman Theorem). *For every minimum k-norm cover $\mathcal{C} = \{C_1, \ldots, C_r\}$, there exists a k-MA which is topologically ordered $\bar{\mathcal{O}} = \{O_1, \ldots, O_k\}$. Moreover, letting $\mathcal{C}' = \{C_i \in \mathcal{C} \mid |C_i| \geq k\}$, it holds that $\mathcal{O}$ and $\mathcal{C}'$ are parallel.*

Fewer Antichains, for Posets. A partially ordered set $P = (V, E)$ is a special type of a DAG for which $TC(P) = P$. All prior algorithms for the k-MA problem assumed that the input graph is a poset. For this class, we provide improved approximation on the number of antichains, by obtaining $(1+\epsilon)k$ antichains that cover $\alpha_k(G)$ vertices. For dense posests and constant ϵ, the running time is almost linear. Specifically, we show:

▶ **Theorem 1.7.** *There is an algorithm k-*MaximumAntichain *that given an n-vertex m-edge poset $P = (V, E)$ and an $\epsilon \in (0, 1)$, computes a set of at most $\ell = (1+\epsilon)k$ vertex-disjoint antichains $\mathcal{A} = \{A_1, \ldots, A_\ell\}$ with $|V(\mathcal{A})| = \alpha_k(P)$. The running time of the algorithm is $O(\sqrt{m} \cdot \alpha_k(G) \cdot n^{o(1)}/\epsilon + n)$.*

This improves over the $O(n^3)$-time algorithm of [8], at the cost of increasing the number of antichains by a factor of $(1+\epsilon)$.

2 Preliminaries

Throughout, we denote the number of edges in the given DAG $G = (V, E)$ by m and the number of vertices by n. Let $TC(G)$ be the transitive closure of G. Let $V = \{v_1, \ldots, v_n\}$. We write $v_i <_G v_j$ if v_i appears before v_j in every topological ordering of G. We write $v_i \leq_G v_j$ if there exists a topological ordering in which v_i does not appear after v_j. For two subsets of vertices $A, B \subseteq G$, we say that $A <_G B$ (resp., $A \leq_G B$) if for every $a \in A$ and $b \in B$ it holds that $a <_G b$ (resp., $a \leq_G b$). When G is clear from the context, we may omit it and simply write $A < B$ and $A \leq B$.

For two ordered vertex subsets $\bar{A} = \{a_1, \ldots, a_\ell\}$ and $\bar{B} = \{b_1, \ldots, b_\ell\}$, $\bar{A} \leq_G \bar{B}$ if for every $i \in \{1, \ldots, \ell\}$, we have $a_i \leq_G b_i$. We say that $u \rightsquigarrow_G v$ if $(u, v) \in TC(G)$. When G is clear from the context, we may omit it and write $v_i < v_j$, $v_i \leq v_j$, $\bar{A} \leq \bar{B}$ and $u \rightsquigarrow v$.

A *chain* is a dipath in $TC(G)$ and an *antichain* is an independent set in $TC(G)$. For a chain $C = [v_1, \ldots, v_\ell]$ and $1 \leq i \leq j \leq \ell$, let $C[v_i, v_j] = C[v_i, v_{i+1}, \ldots, v_j]$. For a subset $V' \subseteq V(C)$, the *first* vertex in V' is the vertex on C that is closest to v_1. For a set of vertex-subsets $\mathcal{S} = \{S_1, \ldots, S_\ell\}$, let $V(\mathcal{S}) = \cup_{S_i \in \mathcal{S}} S_i$. Then $\mathcal{S}$ is a *cover* if $V(\mathcal{S}) = V(G)$. For an integer k, let $\beta_k(G), \alpha_k(G)$ be the maximum number of vertices that can be covered by a collection of k vertex-disjoint chains (resp., antichains) in $TC(G)$. The covering properties by chains and antichains and their relations have been characterized by several celebrated theorems, most notably are the Dilworth's Theorem, Mirsky's Theorem and the Greene-Kleitman Theorem, that we state next.

▶ **Theorem 2.1** (Dilworth's Theorem, [6]). *The size of the largest antichain equals the smallest number of chains into which the vertex-set may be partitioned. I.e.,* $\alpha_1(G) = \min\{h \mid \beta_h(G) = n\}$.

Our algorithms are also based on Mirsky's theorem [15], the dual of Thm. 2.1:

▶ **Theorem 2.2** (Mirsky's Theorem, [15]). *The size of the largest chain equals the smallest number of antichains into which the vertices of a DAG G may be partitioned, i.e.,* $\beta_1(G) = \min\{k \mid \alpha_k(G) = n\}$. *In other words, if $TC(G)$ has no chain of cardinality $k+1$, then its vertices can be partitioned into a union of k antichains (i.e., $TC(G)$ is k-colorable).*

Our algorithmic GK approach uses the notion of minimum *k-norm covers*:

▶ **Definition 2.3.** *The minimum* k-norm cover *problem asks for computing a chain cover $\mathcal{C} = \{C_1, \ldots, C_h\}$ with the minimum k-norm, among all other chain covers for G.*

By the GK theorem, it then holds that the k-norm of the minimum chain cover equals to $\alpha_k(G)$. We use the following GK formulation from Gavril [8]. For every $h \in \{1, \ldots, n\}$, let $\Delta_h(G) = \beta_{h+1}(G) - \beta_h(G)$.

▶ **Theorem 2.4** (Slight Restatement of Theorem 1 from [8],[7]). *(i)* $\Delta_1(G) \geq \Delta_2(G) \geq \ldots \geq \Delta_n(G)$. *(ii) Every h-MC $\mathcal{C} = \{C_1, \ldots, C_h\}$ has a corresponding k-MA $\mathcal{A} = \{A_1, \ldots, A_k\}$ for every $\Delta_{h+1}(G) \leq k \leq \Delta_h(G)$, such that $V(\mathcal{A}) \cup V(\mathcal{C}) = V$ and $|A_i \cap C_j| = 1$ for every $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, h\}$.*

The following theorem summarizes the key properties implied by the Greene-Kleitman Theorem. While some of these properties have been explicitly stated in [8, 7, 17], we provide a proof for completeness:

▶ **Theorem 2.5** (The Extended Greene-Kleitman Theorem). *For a given integer k, let h be such that $\Delta_{h+1}(G) \leq k \leq \Delta_h(G)$ and let $\mathcal{C} = \{C_1, \ldots, C_h\}$ be some h-MC (i.e., $|V(\mathcal{C})| = \beta_h(G)$). Then, the following properties hold:*

1. $\min_{C_i \in \mathcal{C}} |C_i| \geq k$.
2. $\mathcal{C}^* = \mathcal{C} \cup \{\{v\} \mid v \in V \setminus V(\mathcal{C})\}$ *is a minimum k-norm cover.*
3. $TC(G)[V \setminus V(\mathcal{C})]$ *has no chain of length $k+1$ (hence, by Thm. 2.2 it is k-colorable).*

Proof. By Theorem 2.4, there exists a k-MA $\mathcal{A} = \{A_1, \ldots, A_k\}$ such that $V(\mathcal{C}) \cup V(\mathcal{A}) = V$ and $|C_i \cap A_j| = 1$ for every $i \in \{1, \ldots, h\}$ and $j \in \{1, \ldots, k\}$. As each chain intersects with each of the k vertex-disjoint antichains, we have that $|C_i| \geq k$. Next, as $V(\mathcal{C}) \cup V(\mathcal{A}) = V$, and $|V(\mathcal{C}) \cap V(\mathcal{A})| = k \cdot h$, we have that:

$$\alpha_k(G) = |V(\mathcal{A})| = k \cdot h + |V \setminus V(\mathcal{C})| \ . \tag{2.1}$$

By Eq. (1.1), property (1) and Eq. (2.1), the k-norm of $\mathcal{C}^*$ is given by $||\mathcal{C}^*||_k = k \cdot h + |V \setminus V(\mathcal{C})| = \alpha_k(G)$. Hence, by Thm. 1.2, we deduce that $\mathcal{C}^*$ is a minimum k-norm cover. We next prove property (3). Assume towards a contradiction that $TC(G)[V \setminus V(\mathcal{C})]$ contains a chain C' of length $k+1$ and let $\mathcal{C}' = (\mathcal{C}^* \setminus \{\{v\}, v \in C'\}) \cup \{C'\}$. I.e., $\mathcal{C}'$ is obtained by omitting the $k+1$ singleton chains $\{v\}_{v \in C'}$ from $\mathcal{C}^*$ and adding the (single) chain C'. Since $\mathcal{C}'$ has now $h+1$ chains of length at least k, its k-norm is given by $||\mathcal{C}'||_k = ||\mathcal{C}^*||_k - |C'| + k < ||\mathcal{C}^*||_k$, a contradiction to property (2). ◀

Our goal in this paper is to provide time-efficient algorithms for computing a small collection of antichains that maximizes the number of covered vertices. We introduce the following notion of approximation for the k-MA problem:

▶ **Definition 2.6** (Approximate k-MA)**.** *For a given DAG $G = (V, E)$, integers $k, \gamma \geq 1$ a collection of vertex-disjoint antichains $\mathcal{A}$ is γ-**approximate** k-**MA** if $|\mathcal{A}| \leq \gamma \cdot k$ and $|V(\mathcal{A})| \geq \alpha_k(G)$.*

In other words, a γ-approximate k-MA is a set of at most γk antichains covers at least $\alpha_k(G)$ vertices. In a coloring terminology, we color a subset of at least $\alpha_k(G)$ vertices in $TC(G)$ with $\gamma \cdot k$ colors (rather than with the optimal number of k colors).

An alternative plausible approximation variant for the k-MA problem asks for computing exactly k chains that cover at least $\delta \cdot \alpha_k(G)$ vertices, for some $\delta \in (0, 1]$. That is, approximating the number of covered vertices by k chains. This approximation variant for the maximum k-colorable subgraph has been addressed in [14] for general graphs. In this paper, we focus on approximating the number of chains rather than on the number of covered vertices, as any γ-approximate solution for the former can be transformed into a δ-approximation solution for the latter for $\delta = 1/\gamma$.

▶ **Lemma 2.7.** *Any algorithm that computes $\gamma \cdot k$ antichains that cover $\alpha_k(G)$ vertices can be converted into an algorithm that computes k antichains that cover at least $\alpha_k(G)/\gamma$ vertices (with the same running time).*

Minimum-Cost Maximum-Flow Computation. In the *minimum-cost maximum-flow* (MCMF) problem, given is a connected directed graph $G = (V, E, u, c)$ with edges capacities $u \in \mathbb{R}^E_{\geq 0}$ and costs $c \in \mathbb{R}^E$ (which can be negative). The vector $x \in \mathbb{R}^E$ is an s-t *flow* for $s, t \in V$ if $x(e) \in [0, u(e)]$ for all e in E, and for each vertex $v \notin \{s, t\}$ the amount of flow entering v equals the amount of flow leaving v, i.e., $\sum_{e=(a,v)} x(e) = \sum_{e=(v,b)} x(e)$. The *cost* of a flow x is defined by $c(x) = \sum_e c(e)x(e)$. The *value* of an s-t flow, for a source s and a sink t, is the amount of flow leaving s, i.e., $val(x) = \sum_{e=(s,v)} x(e)$ (or equivalently, entering t, i.e., $\sum_{e=(v,t)} x_e$). The objective is to compute a maximum s-t flow of minimum cost denoted by $\sum_{e \in E} c_e x_e$. The following theorem was proven in [19].

▶ **Theorem 2.8** (Theorem 1.1 of [19])**.** *There is a deterministic algorithm that given a m-edge graph with integral vertex demands and edge capacities bounded by U in absolute value, and integral costs bounded by C in absolute value, computes an (exact) minimum cost flow in time $T_{\mathsf{MCMF}}(m) = m^{1+o(1)} \log U \log C$.*

▶ **Definition 2.9.** *For a digraph $G = (V, E)$ and a given valid s-t flow vector $x \in \mathbb{N}^E_{\geq 0}$, a* flow decomposition *is a multiset of s-t dipaths in G given by $\mathcal{Q} = \{P_1, \ldots, P_k\}$, such that for every $e \in E$, it holds that $x(e) = |\{P_i \in \mathcal{Q} \mid e \in P_i\}|$. I.e., $x(e)$ equals the number of paths in the multiset containing e.*

3 From Minimum Chain Cover to Maximum Antichain

As a warm-up, we start by providing an algorithmic version of the Dilworth's theorem [6] in the following sense. We need the following definition in all of our constructions: Sets of chains $\mathcal{C}$ and antichains $\mathcal{A}$ are *parallel* if $|C_i \cap A_j| = 1$ for every $C_i \in \mathcal{C}$ and $A_j \in \mathcal{A}$. For the sake of the subsequent sections, we provide a slightly more general algorithm that will be used in computation of the approximate k-MA solutions. Alg. Chains2Antichain is given as input a DAG G and a chain collection $\mathcal{C}$ which is not necessarily a cover (i.e., possibly $V(\mathcal{C}) \neq V$) but there is a promise that there exists a antichain A that is parallel to $\mathcal{C}$. The algorithm then outputs a parallel antichain A (i.e., with $|C_i \cap A| = 1$ for every $C_i \in \mathcal{C}$). Specifically, for the correctness of Alg. Chains2Antichain to hold it is *not* required that $\mathcal{C}$ is an MCC but rather that there exists a parallel antichain for $\mathcal{C}$. The only requirement for the algorithm to work is that there is at least one antichain that is parallel to $\mathcal{C}$. To compute the MA, we will then apply Alg. Chains2Antichain with the MCC $\mathcal{C}^*$ given as input. The correctness of the algorithm will follow by the Dilworth Theorem which guarantees that $\mathcal{C}^*$ admits a parallel antichain, which is also maximal.

We need the following notation. For every $v \in V(\mathcal{C})$, let $c(v)$ to be the index of the chain in $\mathcal{C}$ which contains v, and $p(v)$ be the position of v in this chain. Furthermore, we denote by $C_i(j)$ the vertex in position j of chain C_i. I.e., For the vertex $v = C_i(j)$ it holds that $c(v) = i$ and $p(v) = j$.

For an input of chain collection $\mathcal{C} = \{C_1, \ldots, C_h\}$, Alg. Chains2Antichain maintains a list of indices in $\{1, \ldots, h\}$, which corresponds to chains whose *representatives* to the antichain haven't been determined yet. The algorithm also maintains a partition of the vertices into red R and non-red $V \setminus R$. The distinction between red and remaining vertices is important only for the sake of obtaining linear running time. The red vertices R are those that *provably* cannot be part of any parallel antichain to $\mathcal{C}$. Initially $R = \emptyset$ and the list consists of all h indices $Q = \{1, \ldots, h\}$. Importantly, the red vertices R cover contiguous regions on the chains in $\mathcal{C}$, in the following manner. If a vertex $v = C_i(j)$ becomes red, then all the vertices $C_i(1), \ldots, C_i(j)$ (i.e., preceding v on the chain C) are marked as red, as well. This defines a *frontier* between R and $V \setminus R$ on each of the chains, represented by a vector $\bar{f} = [f(1), \ldots, f(h)]$. Initially, $\bar{f} = [1, \ldots, 1]$ and as the algorithm proceeds the frontier moves forward until hitting the representatives of the parallel antichain on each of the chains.

An Iteration. Algorithm Chains2Antichain is iterative, and proceeds as long as the list Q is non-empty. The list Q consists of the indices of all chains that are still unresolved (i.e., chains from which no representative is currently taken into the antichain). In each iteration, the algorithm picks some (unresolved) index $i \in Q$ and computes an incoming-BFS tree $T_{in}(v)$ rooted at the vertex $v = C_i(f(i))$ (namely, the frontier of C_i) in the graph $G[V \setminus R]$. All vertices in $T_{in}(v) \setminus \{v\}$ are marked as red (added to R), the frontiers of their respective chains are updated accordingly, and their corresponding chain indices are added to the list Q. The output antichain is obtained by taking the h vertices of the final frontier of each of the h chains. See the formal description below.

Algorithm Chains2Antichain$(G, \mathcal{C})$**:**

Input: DAG $G = (V, E)$, ordered chain collection $\bar{\mathcal{C}} = \{C_1, \ldots, C_h\}$ admitting a parallel antichain.
Output: An ordered parallel antichain $\bar{A} = \{v_1, \ldots, v_h\}$ with $v_j \in C_j$ for $j \in \{1, \ldots, h\}$, excluded nodes $R \subseteq V$.

1. Set $R = \emptyset$, $Q = \{1, \ldots, h\}$ and $f(j) = 1$ for $j \in \{1, \ldots, h\}$.
2. While Q is not empty:
   a. Remove an element i from Q.
   b. Set vertex $v = C_i(f(i))$ (the vertex at the frontier of C_i).
   c. Compute an incoming BFS tree $T_{in}(v)$ rooted at v in $G[V \setminus R]$.
   d. For each vertex $u \in T_{in}(v) \setminus \{v\}$ do:
      i. $R = R \cup \{u\}$.
      ii. If $u \in V(\mathcal{C})$ and $f(c(u)) \leq p(u)$ then:
         - $f(c(u)) \leftarrow p(u) + 1$,
         - $Q \leftarrow Q \cup \{c(u)\}$.
3. Return $\bar{A} = \{C_1(f(1)), \ldots, C_h(f(h))\}$ and R.

Analysis of Algorithm Chains2Antichain. Let ℓ be the number of iterations and let R_τ, Q_τ and $\bar{f}_\tau$ denote the variables R, Q and $\bar{f}$ at the beginning of iteration $\tau \in \{1, \ldots, \ell\}$. Initially, $R_1 = \emptyset, Q_1 = \{1, \ldots, h\}$ and $\bar{f}_1 = \{1, \ldots, 1\}$. Let i_τ be the index taken from the list Q in iteration τ, let $v_\tau = C_{i_\tau}(f_\tau(i_\tau))$ be the vertex at the frontier of the chain C_{i_τ} and $U_\tau = T_{in}(v_\tau) \setminus \{v_\tau\}$, the vertices added to set R in iteration τ.

▶ **Lemma 3.1.** *Algorithm* Chains2Antichain *can be implemented in time* $O(|E(G[R])| + h)$ *where* R *is the output set of red vertices and* h *is the size of the output antichain.*

▶ **Lemma 3.2.** *For every* $\tau \in \{1, \ldots, \ell\}$ *and every* $i \in \{1, \ldots, h\}$*, if* $C_i(j) \in R_\tau$ *then* $j \leq f_\tau(i) - 1$.

Let $\bar{f}^* = \bar{f}_{\tau^*}$ be the final frontier vector, hence the output antichain is given by $\bar{A} = \{v_1, \ldots, v_h\}$ where $v_i = C_i(f_{\tau^*}(i))$ for every $i \in \{1, \ldots, h\}\}$. Let R^* be the output set of red vertices.

▶ **Lemma 3.3.** *No red vertex can be a part of a parallel antichain to* $\mathcal{C}$*. I.e., for every parallel antichain* $\bar{O} = \{o_1, \ldots, o_h\}$ *with* $o_j \in O \cap C_j$ *for every* $j \in \{1, \ldots, h\}$*, it holds that* $O \cap R^* = \emptyset$*. Hence,* $\bar{A} \leq \bar{O}$.

▶ **Lemma 3.4.** *A is a parallel antichain, i.e.,* $|C_i \cap A| = 1$ *for every* $C_i \in \mathcal{C}$.

Proof. We first claim that $f^*(i) \leq |C_i|$ and hence $f^*(i)$ is well-defined for every i. By Lemma 3.2, all vertices in $U'_i = \{C_i(j) \mid j \leq f^*(i) - 1\}$ are marked as red. By Lemma 3.3, the vertices in U'_i cannot be part of a parallel antichain. By the promise that there is a parallel antichain, we conclude that $f^*(i) - 1 \leq |C_i| - 1$, and hence $f^*(i)$ is well-defined.

We next claim that A is an antichain. Assume towards a contradiction otherwise, that there exist $u, v \in A$ such that $u \rightsquigarrow v$ and let P be some u-v path in G. Let $v = C_i(f^*(i))$. Letting τ be the iteration in which v becomes at the frontier of C_i (i.e., $v = C_i(f_{\tau+1}(i))$), we have that i is in the list $Q_{\tau+1}$. Since the algorithm terminates only when the list is empty, there is an iteration $\tau' \geq \tau + 1$, where $i_{\tau'} = i$ and $v_{\tau'} = v$. Since $u = C_{i'}(j' = f^*(i'))$ for some $i' \neq i$, we have that $u \in V \setminus R_{\tau^*}$ (i.e., u never got red). If $V(P) \subseteq (V \setminus R_{\tau'})$, then $u \in U_{\tau'}$

and hence $u \in R_{\tau'+1}$. Contradicting that $u \notin R_{\tau^*}$. It remains to consider the case where $V(P) \setminus R_{\tau'} \neq \emptyset$. Let $z \in P$ be the closest vertex to u on P such that $z \in R_{\tau'}$. This implies that there is an iteration $\tau'' \leq \tau' - 1$ on which z becomes red (and joined the set $R_{\tau''+1}$). In other words, an iteration τ'' where z is in the incoming tree of $v_{\tau''}$ in $G[V \setminus R_{\tau''}]$. By the selection of z, we have that the entire segment $P[u, z]$ is in $V \setminus R_{\tau''}$ and hence, $P[u, z] \subseteq U_{\tau''}$. Concluding that $u \in R_{\tau''+1}$, hence a contradiction. Overall, we conclude A is an antichain of cardinality h, and as $A \subseteq V(\mathcal{C})$, we have that A is parallel to $\mathcal{C}$. ◀

Thm. 1.4 follows by Thm. 1.4 with the time bound for computing MCC by [2].

4 Maximum k Antichains

Recall that $\alpha_k(G)$ is the largest number of vertices that can be covered by a collection of k antichains in a DAG G. In this section we provide time-efficient algorithms for computing (possibly approximate) solutions for k-MA. We start by showing that using a reduction from k-MA to MA by Saks [17], one can compute k-MA in time $T_{\mathsf{MCMF}}(km)$, and hence in time $k \cdot m^{1+o(1)}$. Our subsequent algorithms omit the time dependency in k at the cost of introducing a constant factor of approximation, namely, by outputting a larger collection of antichains covering at least $\alpha_k(G)$ vertices.

▶ **Lemma 4.1.** *Given an n-vertex m-edge DAG $G = (V, E)$ and an integer k, there is an algorithm* $\mathsf{SaksCover}(G, k)$ *that computes a k-MA $\mathcal{A}$ in time $O(T_{\mathsf{MCMF}}(km))$.*

This already improves considerably over the state-of-the-art $O(n^3)$-time algorithm of [8] when $k = o(n)$ and $m = o(n^2)$.

***k*-MA Computation, Proof of Lemma 4.1.** Our algorithm is based on Saks [17] that presented a reduction from k-MA to MA using the notion of k-blowup graph. As we explain later, due to running time considerations, our definition of k-blowup graphs is slightly different than that of Saks [17].

▶ **Definition 4.2.** *Given a DAG $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$, the* k-blowup *of G denoted by $B_k(G) = (V_k, E_k)$ is defined in the following way: $B_k(G)$ contains k copies of G denoted by $G_1, G_2, \ldots, G_k$, where $V(G_i) = \{v_1^i, v_2^i, \ldots, v_n^i\}$ and $E(G_i) = \{(v_j^i, v_\ell^i) \mid (v_j, v_\ell) \in E\}$. Then $B_k(G) = (V_B, E_B)$ where $V_B = \bigcup_i V(G_i)$ and $E(B) = \bigcup_i E(G_i) \cup \{(v_j^i, v_j^{i+1}) \mid v_j \in V\}$.*

The k-blowup graph $B'_k(G)$ of [17] is given by[4] $B'_k(G) = TC(B_k(G))$. The reduction from k-MA to MA using k-blowup graphs is based on the following observation:

▶ **Theorem 4.3.** *Let A be an MA of $B_k(G)$, then $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ is a k-MA for G where each A_i is the antichain induced by A on the subgraph G_i of $B_k(G)$.*

Lemma 4.1 follows by Thm. 4.3 and Thm. 1.4.

[4] In [17], as in all prior algorithms for this problem, the input graph for computing the MA must be a poset as well. This is not needed in our framework, as our MA algorithm works directly on the given DAG (even if it is not a poset).

4.1 2-Approximate Maximum k-Antichains, for General DAGs

The goal of this section is in providing an almost-linear time algorithm for computing a collection of at most $2k$ antichains that cover at least $\alpha_k(G)$ vertices, hence proving Thm. 1.5. Given an n-vertex DAG $G = (V, E)$ and an integer k, the algorithm has two steps. The first step computes a minimum k-norm cover $\mathcal{C}^*$, and second step uses this collection of chains to compute at most $(2k-1)$ vertex-disjoint antichains covering at least $\alpha_k(G)$ vertices. This generalizes the maximum antichain algorithm (for $k = 1$) of Section 3 for any k. As we will see, the first step can be implemented in nearly MCMF time, while the second step – our key technical contribution – can be implemented in *linear* time.

4.1.1 Step I: Min k-Norm Cover in Almost-Linear Time

We show how to compute a chain cover $\mathcal{C}$ with the minimum k-norm. Recall that by the GK theorem it holds that $||\mathcal{C}||_k = \alpha_k(G)$. We show:

▶ **Lemma 4.4.** *There is an algorithm* MinNormCover *that given a DAG $G = (V, E)$ and an integer parameter k computes a minimum k-norm cover $\mathcal{C}$. Moreover, $\mathcal{C}_{\geq k} = \{C_i \in \mathcal{C} \mid |C_i| \geq k\}$ is an h-MC for $h = |\mathcal{C}_{\geq k}|$ and $k \in [\Delta_{h+1}(G), \Delta_h(G)]$. The running time is $\widetilde{O}(T_{\mathsf{MCMF}}(m))$.*

We first observe that by combining the recent works of Cáceres [2] and Kogan and Parter [11], one can compute the h-MC in MCMF time.

▶ **Theorem 4.5** ([2])**.** *Given a DAG $G = (V, E, u, c)$ (on n vertices and m edges) and $s, t \in V$ as an input to the Min-Cost Max-Flow problem where u, c are the capacities (resp., costs) of the edges, whose maximum value is polynomial. Let $\mathcal{P}$ be the collection of paths obtained by computing the flow-decomposition (see Def. 2.9). Then, in time $O(T_{\mathsf{MCMF}}(m))$ one can compute a collection of vertex-disjoint chains $\mathcal{C}$ such that $V(\mathcal{P}) = V(\mathcal{C})$.*

By combining Theorem 4.5 with Theorem 2.5 in [11], we have:

▶ **Lemma 4.6** (Corollary of Theorem 2.5 in [11])**.** *Given n-vertex DAG $G = (V, E)$ and integer h, there is an algorithm* MaximumChainCover *that computes a h-MC $\mathcal{C}$ in $\widetilde{O}(T_{\mathsf{MCMF}}(m))$ time.*

Computing the minimum k-norm cover can be done in MCMF time by using the following observation from Gavril [8] which concludes also the proof of Thm. 4.4.

▶ **Observation 4.7** (Gavril [8])**.** *Given an n-vertex DAG $G = (V, E)$ and an integer $k \in \{1, \ldots, n\}$, one can compute the minimum k-norm cover $\mathcal{C}^*$ by employing $O(\log \omega)$ applications of Alg.* MaximumChainCover*, where $\omega \in \{1, \ldots, n\}$ is the width of the graph.*

4.1.2 Step II: From Min k-Norm Cover to 2-Approximate k-Antichains

This step is given as input a minimum k-norm cover $\mathcal{C}$, and the goal is to obtain $2k$ vertex-disjoint antichains covering $\alpha_k(G)$ vertices, and in *linear* time. The correctness of the algorithm will follow by using the properties of the extended Greene-Kleitman theorem. Towards the end of this section, we prove the following:

▶ **Theorem 4.8** (From Minimum k-Norm Cover to $2k-1$ Antichains)**.** *Given a minimum k-norm chain cover $\mathcal{C}$ of an m-edge DAG $G = (V, E)$, one can compute $(2k-1)$ vertex-disjoint antichains $\mathcal{A} = \{A_1, \ldots, A_{2k-1}\}$ such that $|V(\mathcal{A})| \geq \alpha_k(G)$ in $O(m)$ time.*

Description of Algorithm AntichainCover. The algorithm has two steps, each results in k antichains, leading to a total of $2k-1$ antichains. The algorithm starts by computing the minimum k-norm cover $\mathcal{C}$ by applying the algorithm of Lemma 4.4. We then consider the subset $\mathcal{C}_1$ that consists of all chains of length at least k in $\mathcal{C}$, and denote the remaining vertices by $S = V \setminus V(\mathcal{C}_1)$. The first step computes a collection $\mathcal{A}$ of k antichains that are parallel to $\mathcal{C}_1$. The second step computes a collection $\mathcal{B}$ of k antichains by computing a coloring in the graph $TC(G)[S]$ (without computing the transitive closure itself).

Step 1: Computing a Parallel Antichain Set to the Long Chains. Let $\bar{\mathcal{C}}_1 = \{C_1, \ldots, C_h\}$ be an arbitrary ordering of $\mathcal{C}_1$. The algorithm has k iterations for computing k antichains that are parallel to $\mathcal{C}_1$. Throughout the execution the algorithm also maintains a set of *red* vertices that cannot be part of the future antichains, and the frontiers of each of the chains in $\mathcal{C}_1$. The definition of the red vertices is important only for the sake of obtaining a linear running time. Initially, $R_1 = \emptyset$. Iteration j applies Algorithm Chains2Antichain on the graph $G[V \setminus R_j]$ and with the chain collection $\mathcal{C}_j$ which corresponds to h suffixes of the h chains in $\mathcal{C}$ (computed based on their current frontiers). The output of the iteration is an antichain A_j, and an additional subset R'_j of red vertices. The algorithm then updates the frontier of the h chains, resulting in $\mathcal{C}_{j+1}$. It also defines the set R_{j+1} of red vertices by adding to R_j the sets R'_j and A_j[5]. The output of the algorithm is then given by $\mathcal{A} = \{A_1, \ldots, A_k\}$.

Step 2: Computing a Coloring on Remaining Vertices. The second step computes a $(k-1)$-coloring in the graph $TC(G)[S]$ in $O(m)$ time, as follows. It iterates over all vertices in V according to their topological ordering $v_1, \ldots, v_n$, and compute for each vertex v_i the value $\text{len}(i)$ which corresponds to the maximum number of vertices from S that appear on some incoming path into v_i. The analysis uses the fact that $TC(G)[S]$ is $(k-1)$-colorable to show that $\text{len}(i) \leq k-1$, and that all vertices in S with the same len(.) value form an antichain. In other words, we show that a coloring where each $v_j \in S$ is colored with color $\text{len}(v_j)$ is a valid coloring. For a detailed description see the pseudo-code below.

Algorithm AntichainCover(G, k)**:**

Input: A DAG $G = (V, E)$ and an integer $k \geq 1$.
Output: Set of antichains $\mathcal{A}^*$ with $|\mathcal{A}^*| \leq 2k-1$ and $|V(\mathcal{A}^*)| \geq \alpha_k(G)$.

1. $\mathcal{C} = \text{MinNormCover}(G, k)$
2. Let $\mathcal{C}_1 = \{C \in \mathcal{C} \mid |C| \geq k\}$, $S = V \setminus V(\mathcal{C}_1)$, $h = |\mathcal{C}_1|$ and $R_1 = \emptyset$.
3. Let $\bar{\mathcal{C}}_1 = \{C_{1,1}, \ldots, C_{1,h}\}$ (an arbitrary ordering of $\mathcal{C}_1$).
4. For $j = 1$ to k do:
   a. Set $(\bar{A}_j = \{v_{j,1}, \ldots, v_{j,h}\}, R'_j) = \text{Chains2Antichain}(G[V \setminus R_j], \mathcal{C}_j)$.
   b. For every $\ell \in \{1, \ldots, h\}$, let $C_{j+1,\ell} = C_{j,\ell}[p(v_{j,\ell}) + 1, \ldots]$.
   c. $\mathcal{C}_{j+1} = \{C_{j+1,1}, \ldots, C_{j+1,h}\}$.
   d. $R_{j+1} = R_j \cup R'_j \cup A_j$.
5. Let $\mathcal{A} = \{A_j\}_{j=1}^{k}$.
6. Let $\mathcal{B} = \text{SubsetColoring}(G, S, k)$.
7. Return $\mathcal{A}^* = \mathcal{A} \bigcup \mathcal{B}$.

[5] Note that it might be the case where set R'_j is be empty, this happens when the graph is the union of the h chains, i.e., $G = \bigcup_{i=1}^{h} C_i$. We use the red sets R_j in the time analysis of Lemma 3.1.

Algorithm SubsetColoring(G, S, k)**:**

Input: A DAG G and a subset $S \subseteq V$ such that $TC(G)[S]$ has no chain of length k.
Output: A $(k-1)$-coloring of $TC(G)[S]$ represented by ordered $(k-1)$-antichains $\mathcal{B} = \{B_1, \ldots, B_{k-1}\}$.

1. $\{v_1, v_2, \ldots, v_n\} \leftarrow$ TopologicalOrder(G).
2. Set maxlen $= 0$.
3. For $i = 1$ to n do the following:
   a. If $v_i \in S$ then set $\text{len}(i) = 1$, otherwise set $\text{len}(i) = 0$.
   b. Set $\text{len}(i) = \text{len}(i) + \max_{\{j \mid (v_j, v_i) \in E\}} \text{len}(j)$.
   c. Set maxlen $= \max\{\text{maxlen}, \text{len}(i)\}$.
4. For each $1 \leq i \leq$ maxlen set $B_i = \{v_j \mid \text{len}(j) = i \text{ and } v_j \in S\}$.
5. Return $\mathcal{B} = \{B_i\}_{i \in \{1, \ldots, \text{maxlen}\}}$.

Analysis of Algorithm AntichainCover. For a given DAG $G = (V, E)$, let $V = \{v_1, \ldots, v_n\}$.To prove the correctness of Algorithm AntichainCover, one needs to show that the desired promise holds for $\mathcal{C}_j$ and $G_j = G[V \setminus R_j]$ for every $j \in \{1, \ldots, k\}$. I.e., that there exists an antichain A_j in $TC(G_j)$ that is *parallel* to $\mathcal{C}_j$. To prove this claim, we observe the ordered variant of the GK theorem (see Lemma 1.6) implied by the GK Theorem and Mirsky's Theorem.

Proof of Lemma 1.6. Let $\mathcal{A}$ be a k-MA which exists by Theorem 1.2. Hence, $|V(\mathcal{A})| = \alpha_k(G)$. We show that one can color the graph $TC(G)[V(\mathcal{A})]$ with k ordered color sets $\bar{\mathcal{O}} = \{O_1, \ldots, O_k\}$ which are topologically ordered, i.e., $O_1 < O_2 < \ldots < O_k$. By Mirsky's theorem (Theorem 2.2), the longest chain in $TC(G)[V(\mathcal{A})]$ has length at most k. Define a color class O_i for $i \in \{1, \ldots, k\}$ as the set of all vertices $u \in V(\mathcal{A})$ whose longest path in $TC(G)[V(\mathcal{A})]$ from any node to u is of length i. Clearly, $V(\mathcal{O}) = V(\mathcal{A})$. We now claim that $\bar{\mathcal{O}}$ is topologically ordered by showing that there is *no* edge $(u, z) \in TC(G)[V(\mathcal{A})]$ for any $u \in O_j$ and $z \in O_i$ for any $i \leq j$. By the definition of O_j, O_i, the longest path in $TC(G)[V(\mathcal{A})]$ ending at u, z is of length j (resp., i). The existence of an edge (u, z) provides a path of length $j + 1 > i$ ending at z, leading to a contradiction.

We next show that $\mathcal{O}$ and $\mathcal{C}'$ are parallel. Suppose towards a contradiction that there is a $O_j \in \mathcal{O}$ and $C_i \in \mathcal{C}'$ such that $C_i \cap O_j = \emptyset$. Letting $V' = V(\mathcal{C}')$ and $h = |\mathcal{C}'|$, since each chain in $\mathcal{C}'$ has at most k mutual vertices with $V(\mathcal{O})$, we have that:

$$\alpha_k(G) = |V(\mathcal{O})| = |V(\mathcal{O}) \cap (V \setminus V')| + |V(\mathcal{O}) \cap V'| \leq |V \setminus V'| + k \cdot h - 1 < ||\mathcal{C}||_k = \alpha_k(G) \ ,$$

leading to a contradiction. The claim follows. ◀

Let $\bar{\mathcal{O}} = \{O_1, \ldots, O_k\}$ be a topologically ordered k-MA which is parallel to the chains $\mathcal{C}_1$, whose existence is guaranteed by Lemma 1.6. The next lemma shows that in each application $j \in \{1, \ldots, k\}$ of Alg. Chains2Antichain by Alg. AntichainCover, it holds that there exists a collection of $(k - j + 1)$ topologically ordered antichains $\mathcal{O}_j = \{O_j, \ldots, O_k\}$ that are parallel to $\mathcal{C}_j$.

▶ **Lemma 4.9.** *For every $j \in \{1, \ldots, k\}$, it holds that (i) $A_j \cap A_{j-1} = \emptyset$ and (ii) $\bar{\mathcal{O}}_j = \{O_j, \ldots, O_k\}$ is parallel to $\bar{\mathcal{C}}_j = \{C_{j,1}, \ldots, C_{j,h}\}$.*

Proof. Property (i) follows by the fact that at the end of phase j, the computed antichain A_j is added to the set of red vertices R_{j+1}. Since $V(\mathcal{C}_{j+1}) \cap R_{j+1} = \emptyset$ and $A_{j+1} \subset V(\mathcal{C}_{j+1})$, we have that $A_j \cap A_{j+1} = \emptyset$ for every $j \in \{1, \ldots, k\}$.

We next turn to prove claim (ii). Assume by induction on $j \geq 1$, that at the beginning of phase j, there are $k - j + 1$ topologically ordered antichains $\bar{\mathcal{O}}_j = \{O_j, \ldots, O_k\}$ that are parallel to $\bar{\mathcal{C}}_j = \{C_{j,1}, \ldots, C_{j,h}\}$. By Lemma 3.4 it then holds that the output antichain of phase j, namely, $\bar{A}_j$, is parallel to $\bar{\mathcal{C}}_j$, where $\bar{A}_j = \{v_{j,1}, \ldots, v_{j,h}\}$ consists of the vertices of A_j, where $v_{j,\ell} \in C_{j,\ell}$. Moreover, by Lemma 3.3 it also holds that $A_j \leq O_j$, and since $\bar{\mathcal{O}}_i$ is topologically ordered, it also holds that $O_j < \ldots < O_k$. Since $O_j < O_{j+1}$, we have that $V(\mathcal{O}_{j+1}) \subseteq V(\mathcal{C}_{j+1})$ where $\mathcal{O}_{j+1} = \{O_{j+1}, \ldots, O_k\}$. ◀

▶ **Corollary 4.10.** *The set $\mathcal{A} = \{A_1, \ldots, A_k\}$ is parallel to $\mathcal{C}'$. Hence, $|V(\mathcal{A})| \geq |\mathcal{C}'| \cdot k$.*

We next turn to consider the second set of antichains $\mathcal{B}$ obtained by Alg. SubsetColoring which colors the vertices in S by $k-1$ colors, where $B_i \subseteq S$ are the vertices belonging to the i^{th} color-class.

▶ **Lemma 4.11.** *Given an m-edge DAG G and a subset $S \subseteq V$ such that $TC(G)[S]$ has no chain of length $\geq k$, Alg.* SubsetColoring *computes a topologically ordered coloring $\bar{\mathcal{B}} = \{B_1, \ldots, B_\ell\}$ for $\ell \leq k-1$ such that $V(\mathcal{B}) = S$ and each B_i is an antichain. The running time of the algorithm is $O(m)$.*

Proof. We claim by induction on $i \geq 1$ that $\text{len}(i)$ equals to the length of the longest path in $TC(G)[S]$ ending at v_i. The base of the induction holds as $\text{len}(v_1) = 1$ if $v_1 \in S$ and 0 otherwise. Assume that the claim holds up to i and consider $\text{len}(i+1)$. Let P be the longest path in $TC(G)[S]$ ending at v_{i+1} and let (v_j, v_{i+1}) be the last edge of P. By the topological ordering, we have that $j < i$. By the induction assumption for j, we have that $\text{len}(j) = |P| - 1$, and therefore $\text{len}(i) = |P|$ as desired. The induction step holds. We next claim that there no edge $(v_a, v_b) \in TC(G)$ for $v_a \in B_i$ and $v_b \in B_j$ for any $j \leq i$. Assume towards a contradiction otherwise, we get that $j = \text{len}(b) \geq \text{len}(a) + 1 = j + 1$, leading to a contradiction. Note that by the promise $TC(G)[S]$ has no chain of length k, hence $\mathsf{maxlen} \leq k-1$ for every i. We conclude that $V(\mathcal{B}) = S$ and that $\mathcal{B}$ is a legal coloring. Finally, we consider the running time, the algorithm has n iterations where iteration i is implemented in $\deg_{in}(v_i, E) = |\{v_j \mid (v_j, v_i) \in E(G)\}|$, hence overall the algorithm is implemented in $O(m)$ time. ◀

We are now ready to complete the correctness of Alg. AntichainCover and bound its running time which is dominated by the computation of the minimum k-norm cover. Thm. 1.5 follows by combining Lemma 4.12, Obs. 4.7 and Lemma 2.7.

▶ **Lemma 4.12.** *The output of Alg.* AntichainCover(G, k) *given by $\mathcal{A} \cup \mathcal{B}$ is a 2-approximate k-MA with $|\mathcal{A}| + |\mathcal{B}| \leq 2k - 1$. The running of the algorithm is $\widetilde{O}(T_{\mathsf{MCMF}}(m))$.*

References

1 Ron Aharoni and Irith Ben-Arroyo Hartman. On greene-kleitman's theorem for general digraphs. *Discret. Math.*, 120(1-3):13–24, 1993.

2 Manuel Cáceres. Minimum chain cover in almost linear time. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 31:1–31:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.31.

3 Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 359–376. SIAM, 2022.

4 Glenn G. Chappell. Polyunsaturated posets and graphs and the greene-kleitman theorem. *Discret. Math.*, 257(2-3):329–340, 2002.

5 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022.

6 RP Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, pages 161–166, 1950.

7 András Frank. On chain and antichain families of a partially ordered set. *J. Comb. Theory, Ser. B*, 29(2):176–184, 1980.

8 Fănică Gavril. Algorithms for maximum k-colorings and k-coverings of transitive graphs. *Networks*, 17(4):465–470, 1987.

9 Curtis Greene and Daniel J Kleitman. The structure of sperner k-families. *Journal of Combinatorial Theory, Series A*, 20(1):41–68, 1976.

10 Alan J. Hoffman and D. E. Schwartz. On partitions of a partially ordered set. *J. Comb. Theory, Ser. B*, 23(1):3–13, 1977.

11 Shimon Kogan and Merav Parter. Beating matrix multiplication for $n^{1/3}$-directed shortcuts. In *The 49th EATCS International Colloquium on Automata, Languages and Programming ICALP 2022*, 2022. Full version available at `https://www.weizmann.ac.il/math/parter/sites/math.parter/files/uploads/main-lipics-full-version_3.pdf`.

12 Shimon Kogan and Merav Parter. Faster and unified algorithms for diameter reducing shortcuts and minimum chain covers. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 212–239. SIAM, 2023.

13 Nathan Linial. Extending the greene-kleitman theorem to directed graphs. *J. Comb. Theory, Ser. A*, 30(3):331–334, 1981.

14 Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. *In Svante Carlsson Andrzej Lingas, Rolf G. Karlsson, editor, Automata, Languages and Programming, 20th International Colloquium, volume 700 of Lecture Notes in Computer Science*, pages 40–51, July 1993.

15 Leon Mirsky. A dual of dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971.

16 Giri Narasimhan. The maximum k-colorable subgraph problem. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1989.

17 Michael Saks. A short proof of the existence of k-saturated partitions of partially ordered sets. *Advances in Mathematics*, 33(3):207–211, 1979.

18 Michael E. Saks. Kleitman and combinatorics. *Discret. Math.*, 257(2-3):225–247, 2002.

19 Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 503–514. IEEE, 2023.

20 Douglas B. West. "Poly-unsaturated" posets: The Greene-Kleitman theorem is best possible. *J. Comb. Theory, Ser. A*, 41(1):105–116, 1986.