Exponential-Time Approximation (Schemes) for Vertex-Ordering Problems

Matthias Bentert
University of Bergen, Norway

Fedor V. Fomin
University of Bergen, Norway

Tanmay Inamdar
Indian Institute of Technology Jodhpur, India

Saket Saurabh
The Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

Abstract

In this paper, we begin the exploration of vertex-ordering problems through the lens of exponential-time approximation algorithms. In particular, we ask the following question: Can we simultaneously beat the running times of the fastest known (exponential-time) exact algorithms and the best known approximation factors that can be achieved in polynomial time? Following the recent research initiated by Esmer et al. (ESA 2022, IPEC 2023, SODA 2024) on vertex-subset problems, and by Inamdar et al. (ITCS 2024) on graph-partitioning problems, we focus on vertex-ordering problems. In particular, we give positive results for Feedback Arc Set, Optimal Linear Arrangement, Cutwidth, and Pathwidth. Most of our algorithms build upon a novel "balanced-cut" approach – which is our main conceptual contribution. This allows us to solve various problems in very general settings allowing for directed and arc-weighted input graphs. Our main technical contribution is a $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ for (weighted) Feedback Arc Set in $O^*((2-\delta_\varepsilon)^n)$ time, where $\delta_\varepsilon > 0$ is a constant only depending on ε.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Feedback Arc Set, Cutwidth, Optimal Linear Arrangement, Pathwidth

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.15

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
Fedor V. Fomin: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
Tanmay Inamdar: Supported by IITJ Research Initiation Grant (grant number I/RIG/TNI/20240072).
Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416); and he also acknowledges the support of Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

Many NP-hard problems admit algorithms running in time $O^*(2^n)$[1]. Moreover, for fundamental problems like Satisfiability or Set Cover, there are plausible conjectures, namely Strong Exponential Time Hypothesis (SETH) or the Set Cover Conjecture, respectively,

[1] Throughout the introduction, n will have the "natural" meaning, e.g., number of variables/vertices/elements, and the O^*-notation hides polynomial factors. See Section 2 for more details about the latter.

which state that these problems do not admit algorithms running in time $O^*((2-\varepsilon)^n)$ for any $\varepsilon > 0$. Although these conjectures have been used as a source of analogous running-time lower bound for other problems, many graph problems are not covered by such results. Indeed, quite a few long-standing exponential-time barriers have finally been overcome in recent years [33, 37, 34], giving new life to the field of *moderately exponential-time algorithms* [22].

Improving upon the trivial $O^*(2^n)$-time algorithm for "vertex-subset problems" on graphs, Fomin et al. [21] designed a general framework called *monotone local search* (MLS). This framework enables the creation of state-of-the-art exact exponential algorithms for a wide range of such problems by leveraging the corresponding parameterized algorithms as black-box components. On the other hand, for graph partitioning and vertex-ordering problems, even the $O^*(2^n)$-time algorithm does not follow from a trivial brute force of the search space. Nevertheless, many such problems do admit $O^*(2^n)$-time algorithms via advanced techniques such as Held-Karp-style dynamic programming [23, 6] or speeding up the natural dynamic program via fast subset convolution [26]. For these latter two classes of problems, beating the $O^*(2^n)$ barrier has proven to be challenging, with the notable exception of k-Cut for which Lokshtanov et al. [31] designed an $O^*((2-\varepsilon)^n)$-time algorithm for some fixed $\varepsilon > 0$. Due to the lack of any progress for many of these intensively studied vertex-ordering problems, we think it is the right time to relax the question just slightly in the following way.

> Can we show or exclude $(1+\varepsilon)$-approximations for vertex-ordering problems that run in $O((2-\delta)^n)$ time?

We show that this question can be answered affirmatively for Feedback Arc Set, but we were not able to achieve the same for any other problem. However, the methods we develop for Feedback Arc Set allow us to answer another natural (weaker) relaxation:

> Can we simultaneously beat the running times of the fastest known (exponential-time) exact algorithms and the best known approximation factors that can be achieved in polynomial-time for vertex-ordering problems?

Given the difficulty in surmounting the current-best exact exponential-time algorithms, there has been a growing research interest in answering the latter relaxation. This is also justified given the fact that most of the problems are known to be APX-hard – they do not admit arbitrarily good approximations. In some cases, problems do not even admit constant-factor approximations in polynomial time assuming $P \neq NP$. To the best of our knowledge, the first exponential-time approximation algorithms were given by Williams [36] who showed how to generalize $O^*(2^{\omega n/3})$-time algorithms for (unweighted) Max Cut and Max 2-SAT to $(1+\varepsilon)$-approximations for the weighted variants using "rounding tricks"; here ω denotes the fast matrix multiplication constant. Quite recently, Alman et al. [1] showed that Max Sat can also be approximated faster than $O^*(2^n)$, that is, they showed that for each $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation in $O^*((2-\delta_\varepsilon)^n)$ time where $\delta_\varepsilon > 0$ is a constant only depending on ε. Moreover, the monotone local search framework of Fomin et al. [21] was extended to give exponential-time approximation schemes for vertex-subset problems in the works of Esmer et al. [15, 16, 17]. Even more recently, Inamdar et al. [26] gave a framework for designing exponential-time approximation schemes for graph partitioning (i.e., cut) problems and a few more examples for specific problems are known [8, 9, 4]. However, none of these techniques seem applicable to vertex-ordering problems, where the objective is to find an ordering of vertices that satisfies some polynomial-time satisfiable predicate, such as Feedback Arc Set, Optimal Linear Arrangement, Cutwidth, and Pathwidth. For all of the above except Pathwidth, the $O^*(2^n)$-time dynamic programming remains the

Table 1 An overview of our results for unweighted input graphs. For each problem in the left column, we show an algorithm with approximation factor in the middle column and the running time is shown in the right column. The value $\omega < 2.373$ is the fast matrix multiplication exponent and $\delta > 0$ is a constant that depends on ε.

problem	approximation factor	running time
Feedback Arc Set	$1+\varepsilon$	$O^*((2-\delta)^n)$
Directed Cutwidth	2	$O^*(2^{\frac{\omega n}{3}})$
Optimal Linear Arrangement	$1.5+\varepsilon$	$O^*((2-\delta)^n)$
Directed Optimal Linear Arrangement	$2+\varepsilon$	$O^*((2-\delta)^n)$
Directed Pathwidth	2	$O^*(1.66^n)$

best known algorithm–even including constant-factor approximations. Moreover, the question whether e.g., Feedback Arc Set or Cutwidth can be solved faster than $O^*(2^n)$ have been explicit open problems for a long time [22, 24, 25, 12]. This reinforces our belief that it is time to study exponential-time approximation algorithms for vertex-ordering problems. The only example of such a study is in the case of (directed or undirected) bandwidth [13, 28]. Our results are summarized in Table 1. We believe that working on these vertex-ordering problems from a new angle will further enhance our understanding of them and we hope that there might be some interesting connections to find that use ideas from the vast knowledge acquired on these problems both from the world of polynomial-time approximations and from the world of exact exponential-time algorithms.

In the rest of this section, we survey a few different vertex-ordering problems and previous work related to them. Afterwards, we give an overview of our results and a high-level intuition of our main conceptual contribution.

Problem Definitions and Related Work

We now introduce the different problems used in this paper and briefly state relevant related work for each of them.

We start with a directed version of Minimum Cut with an additional size constraint.

Directed Minimum $(k, n-k)$-Cut

Input: A directed graph G and an integer k.

Task: Find a set L of exactly k vertices such that the number of arcs from $V \setminus L$ to L is minimized.

The undirected version of this problem was introduced by Bonnet et al. [7], who showed fixed-parameter tractability of the problem when parameterized by $k + \mathrm{OPT}$, where OPT is the minimum number of arcs from $V \setminus L$ to L for any set L of size k. We are not aware of any mentioning of the directed problem anywhere. We also study a weighted generalization where we are given an additional non-negative arc-weight function w and instead of minimizing the number of arcs from $V \setminus L$ to L, we instead minimize the weight of the arc set from $V \setminus L$ to L. Directed Minimum $(k, n-k)$-Cut will be at the heart of most of our algorithms.

We continue with Feedback Arc Set. It can be defined both in terms of a subset of arcs of an input graph and as a vertex-ordering problem. The former definition asks for a given directed graph $G = (V, A)$ to find a minimum-size set of arcs whose removal turns G into a directed acyclic graph (DAG). We give the definition as a vertex-ordering problem as this will turn out to be more useful for us later.

Feedback Arc Set
Input: A directed graph $G = (V, A)$.
Task: Find an ordering π of V minimizing the number of backward arcs with respect to π.

Feedback Arc Set can be solved in $O(2^n n^2)$ time by the Held-Karp algorithm [23]. The weighted version of this problem asks for an ordering such that the sum of weights of all backward arcs is minimized. The best known polynomial-time approximaiton algorithm (for both the weighted and unweighted variants) achieves an approximation factor of $O(\log(n) \log\log(n))$ [18] and any approximation better than 1.36 in polynomial time would refute $P \neq NP$ [29, 14]. As stated above, we achieve a $(1+\varepsilon)$-approximation for Feedback Arc Set faster than $O(2^n)$ for any $\varepsilon > 0$. We mention that some of the methods used for this result can also be applied to other vertex-ordering problems (to obtain constant-factor approximations). We list a few such problems in the following.

In Optimal Linear Arrangement (also known as Minimum Linear Arrangement), one is asked for an ordering of a graph that minimizes the sum of stretches of all (backward) arcs. Equivalently, the problem asks for an ordering minimizing the sum of cut sizes over all positions. In the following, we denote by cut^i_π for an ordering π of the vertices and an integer i, the set of all edges (or arcs) from a vertex after position i to a vertex at position at most i.

Optimal Linear Arrangement
Input: An directed or undirected graph G with vertex set V.
Task: Find an ordering π of V minimizing $\sum_{i=1}^{n-1} |\,\mathrm{cut}^i_\pi\,|$.

In the weighted setting, we want to minimize $\sum_{i=1}^{n-1} \sum_{e \in \mathrm{cut}^i_\pi} w(e)$. Optimal Linear Arrangement can be solved in $O^*(2^n)$ time [6] and the best known polynomial-time algorithm achieves an approximation factor in $O(\sqrt{\log(n)} \log\log(n))$ [20, 10]. It remains NP-hard even when restricted to interval graphs but there it can be 2-approximated in polynomial time [11].

In the problem Directed Cutwidth[2], we are looking for an ordering that minimizes the maximum size of any cut^i_π. Formally, it is defined as follows.

Directed Cutwidth
Input: A directed graph $G = (V, A)$.
Task: Find an ordering π of V minimizing $\max_{i \in [n-1]} |\,\mathrm{cut}^i_\pi\,|$.

In the weighted setting, we want to minimize the weight $\max_{i \in [n-1]} \sum_{e \in \mathrm{cut}^i_\pi} w(e)$ instead of the cardinality. Directed Cutwidth can be solved in $O(2^n n^2)$ time and the weighted version can be solved in $O(2^n n^2 \log(w_{\max}))$ time by the Held-Karp algorithm [23]. The best known approximation factor achieved by a polynomial-time algorithm is $O(\log^{1.5}(n))$ [3]. A quantum algorithm running in $O(1.817^n)$ time is also known [2] and if the input graph is undirected, then the problem admits a $\log^{1+o(1)}(n)$-approximation [5].

Finally, we study Directed Pathwidth. It is a directed generalization of Pathwidth which itself is variation of treewidth where the tree decomposition is restricted to being a path. It is known that Pathwidth has an equivalent definition in terms of vertex orderings

[2] We mention in passing that there is some inconsistency in the existing literature regarding the name Directed Cutwidth. An alternative definition takes a directed acyclic graph as input and searches for a topological ordering of the vertices minimizing the (undirected) cutwidth.

and this remains true also for the directed variant [30]. For a position i, consider the number of vertices v that appear before position i that have in-neighbors that appear after position i. The equivalent definition of Directed Pathwidth asks for an ordering that minimizes the maximum quantity over all positions. The formal definition in terms of vertex orderings is as follows.

Directed Pathwidth
Input: A directed graph $G = (V, A)$.
Task: Find an ordering π of V minimizing

$$\max_{i \in [n-1]} |\{v \in V \mid \pi(v) \leq i \wedge (\exists u \in V.\ \pi(u) > i \wedge (u, v) \in A)\}|.$$

The fastest exact algorithm for Directed Pathwidth runs in $O(1.89^n)$ time [30] and the best known approximation factor of a polynomial-time algorithm is $O(\log^{1.5}(n))$ [19]. The undirected pathwidth can be $\log^{1+o(1)}(n)$-approximated in polynomial time [5].

Overview

In this work, we initiate the study of exponential-time approximation algorithms for vertex-ordering problems. A recurring theme in our algorithms is a "balanced-cut" strategy, which results in a running time that is strictly faster than $O^*(2^n)$ and which we consider our main conceptual contribution. We briefly describe the main idea here. We first show how to find a partition of the set of vertices in a graph into two sets L and R of given sizes such that the number of arcs from R to L is (approximately) minimized, that is, we solve Directed Minimum $(k, n-k)$-Cut.[3] Depending on the specific problem at hand, we then either solve the two instances induced by these sets of vertices exactly or approximately. Finally, we show how to combine the two partial solutions with the computed cut between R and L. For Feedback Arc Set, we design an additional "self-improving" $(1+\varepsilon)$-approximation algorithm by combining the above approach with a novel bootstrapping strategy.

The rest of this work is structured as follows. In Section 2, we introduce notation and concepts used in this work. We show our results for unweighted problems in Section 3 and for weighted problems in Section 4. We conclude with Section 5.

2 Preliminaries

For a positive integer $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \ldots, n\}$. We use standard graph-theoretic notation. In particular, for a given graph $G = (V, E)$ and a set $V' \subseteq V$ of vertices, we use $G[V']$ to denote the subgraph of G induced by V'. To avoid confusion, we will use the term *edges* for undirected graphs and the term *arcs* for directed graphs. Moreover, we will refer to the edge set of an undirected input graph by E and the arc set of a directed input graph by A. All graphs in this work are simple, that is, they do not contain self-loops or parallel edges/arcs (but we allow in the directed case both the arc (u, v) and the arc (v, u) to be present at the same time for a pair of vertices $u \neq v$). We always denote the number of vertices in the input graph by n and the number of edges/arcs by m. We say that a graph is edge-weighted (respectively arc-weighted) if we are given an additional weight function $w \colon E \to \mathbb{N}$ ($w \colon A \to \mathbb{N}$). The weight of an edge $e \in E$ is then $w(e)$ and the weight of a set $E' \subseteq E$ of edges is the sum of edge weights of edges in E.

[3] We mention that we do not always just consider a single partition. Sometimes, we iterate over many such partitions and return the best solution over all considered partitions.

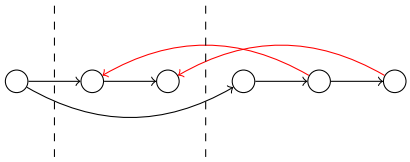

Figure 1 An example for cut_π in the directed setting. The vertices are ordered by some ordering π from left to right. The cuts cut_π^1 and cut_π^3 are depicted by dashed lines. The cut cut_π^1 is empty and the cut cut_π^3 consists of all the red arcs. The stretch of both red arcs is 3.

A *ordering* π of the vertices of a graph G with vertex set V is a bijection between V and $|V|$. For a directed graph $G = (V, A)$ and an ordering π of V, we say that an arc (u, v) is forward with respect to π if $\pi(u) < \pi(v)$ and the arc is backward with respect to π otherwise. The stretch of an edge $\{u, v\}$ (or an arc (u, v)) is $|\pi(u) - \pi(v)|$. For an integer $i \in [n]$, we say the cut at position i with respect to π is $\mathrm{cut}_\pi^i = \{\{u, v\} \in E \mid \pi(u) > i \wedge \pi(v) \leq i\}$. In the directed case, we use the definition $\mathrm{cut}_\pi^i = \{(u, v) \in A \mid \pi(u) > i \wedge \pi(v) \leq i\}$. See Figure 1 for an example.

We assume familiarity with the Bachmann–Landau notation (also known as big-O notation) and we use O^* to hide polynomial factors in the input size. We note, that this on the one hand hides polynomial factors in $\log(w_{\max})$, where $w_{\max}$ is the largest weight in the input graph. On the other hand, $O^*(\alpha^n) \subseteq O((\alpha + \varepsilon)^n)$ for any $\alpha \geq 1$ and any $\varepsilon > 0$.

3 Unweighted Problems

In this section, we present our algorithms for unweighted input graphs. We present all algorithms for the directed case but mention that it is straight-forward to adapt all of these algorithms to the undirected case. Most of our results are based on the following faster algorithm for Directed Minimum $(k, n-k)$-Cut which is a generalization of a similar algorithm for Maximum Cut by Williams [36]. Here and in the following, $\omega < 2.373$ is the matrix-multiplication exponent.

▶ **Proposition 1.** *Directed Minimum $(k, n-k)$-Cut can be solved in $O^*(\min(2^{\frac{\omega n}{3}}, n^{\omega k}))$ time.*

Proof. Let $(G = (V, A), k)$ be an input instance of Directed Minimum $(k, n-k)$-Cut. We partition V arbitrarily into three sets V_1, V_2, and V_3 of (roughly) equal size. Next, we guess in $O(n^2)$ time how many vertices of V_1, V_2, and V_3 belong to an optimal solution L. Let these numbers be k_1, k_2 and $k_3 = k - k_1 - k_2$. Next, we build a graph H as follows. For each $i \in [3]$, we introduce a vertex v_j^i for each subset $V_j^i \subseteq V_i$ with $|V_j^i| = k_i$. Let $\delta(v_j^i) = |\{(u, w) \in A \mid u \in V_i \setminus V_j^i \wedge w \in V_j^i\}|$ be the number of arcs from $V_i \setminus V_j^i$ to V_j^i. Note that H contains $O(\min(2^{n/3}, \binom{n/3}{k}))$ vertices. For each pair of vertices $v_j^i, v_{j'}^{i'}$ with $i \neq i'$, we add an edge between them. We set the weight of this edge to

$$|\{(u, w) \in A \mid u \in V_i \setminus V_j^i \wedge w \in V_{j'}^{i'}\}|$$
$$+ |\{(u, w) \in A \mid u \in V_{i'} \setminus V_{j'}^{i'} \wedge w \in V_j^i\}| + \frac{\delta(v_j^i) + \delta(v_{j'}^{i'})}{2}.$$

Note that each edge weight is at most n^2. Next, for each pair $i \neq i'$, we guess the weight of the edge $\{v_j^i, v_{j'}^{i'}\}$ in a minimum-weight triangle in H (without guessing the indices j and j'). Note that there are n^6 possible guesses. For each guess, we compute in time linear in the size of H

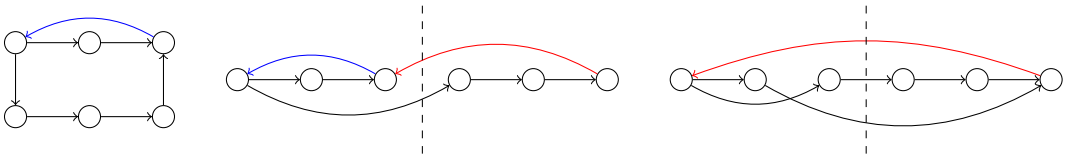

Figure 2 An example of Feedback Arc Set. The input graph is given on the left and the middle/right show two optimal solutions for Directed Minimum $(k, n-k)$-Cut with $k = 3$ (red arcs). The optimal solution to Feedback Arc Set is shown in blue (and coincides with the red arc on the right). If we use the cut in the middle picture, then we can only find a 2-approximation while the cut on the right leads to an optimal solution.

a subgraph which only contains the edges corresponding to the current guess. We then check whether this subgraph contains a triangle in $O^*((\min(2^{\frac{n}{3}}, \binom{n/3}{k}))^\omega) = O^*(\min(2^{\frac{\omega n}{3}}, n^{\omega k}))$ time [27]. The algorithm also finds a triangle if one exists. After going through all possible guesses, we report the triangle of minimum weight. Note that this weight corresponds by construction to a minimum directed $(k, n-k)$-cut such that $|L \cap V_i| = k_i$ for all $i \in [3]$. Since we iterate over all possible choices of k_i, we always find a minimum-weight directed $(k, n-k)$-cut. The total running time is in $O^*(\min(2^{\frac{\omega n}{3}}, n^{\omega k}))$. ◀

3.1 Feedback Arc Set

We next show how to use Proposition 1 to design a $(1+\varepsilon)$-approximation for Feedback Arc Set for any $\varepsilon > 0$ that is faster than the $O^*(2^n)$-time Held-Karp algorithm. We first show how to find a 2-approximation in $O^*(2^{\frac{\omega n}{3}})$ time and then show how to use a $(1+\frac{1}{k})$-approximation to find a $(1+\frac{1}{k+1})$-approximation.

▶ **Lemma 2.** *Feedback Arc Set can be 2-approximated in $O^*(2^{\frac{\omega n}{3}})$ time.*

Proof. Let $G = (V, A)$ be an input graph for Feedback Arc Set. We first compute an optimal solution $L \subseteq V$ of the instance $(G, \lfloor \frac{n}{2} \rfloor)$ of Directed Minimum $(k, n-k)$-Cut in $O^*(2^{\frac{\omega n}{3}})$ time using Proposition 1. Observe that $|L| = \lfloor \frac{n}{2} \rfloor$, and that the number of arcs from $V \setminus L$ to L is a lower bound for the size of a minimum feedback arc set in G. Let this set of arcs be A' and let the set of arcs from L to $V \setminus L$ be A''. Next, we solve the two instances $G[L]$ and $G[V \setminus L]$ of Feedback Arc Set optimally in $O^*(2^{\frac{n}{2}})$ time [23]. See Figure 2 for an example. Since a feedback arc set in a graph is also a feedback arc set in any subgraph, it holds that the optimal solutions in $G[L]$ and $G[V \setminus L]$ are combined a lower bound for the size of a feedback arc set in G. Hence, placing the vertices of L in the computed order before the vertices of $V \setminus L$ in the computed order yields a 2-approximation as each arc in A' contributes one and the arcs in A'' do not contribute anything. The running time is in $O^*(2^{\frac{\omega n}{3}} + 2^{\frac{n}{2}}) = O^*(2^{\frac{\omega n}{3}})$. ◀

We next show how to use any constant-factor approximation for Feedback Arc Set that is faster than $O^*(2^n)$ to compute a better constant factor approximation that is also faster than $O^*(2^n)$. This "boosting technique" results in the following approximation scheme.

▶ **Theorem 3.** *For each constant $\varepsilon > 0$, there is a $\delta_\varepsilon > 0$ such that Feedback Arc Set can be $(1+\varepsilon)$-approximated in $O((2-\delta_\varepsilon)^n)$ time.*

Proof. We prove this statement by showing via induction that for each $k \geq 1$, there is a $\delta_k > 0$ such that Feedback Arc Set can be $(1+\frac{1}{k})$-approximated in $O((2-\delta_k)^n)$ time. Since any constant $\varepsilon > 0$ is larger than $\frac{1}{k}$ for some constant value of k, this will prove the

theorem. Note that the base case $k = 1$ is proven by Lemma 2. It remains to show the induction step, that is, assuming that the statement is true for some value of k, we need to show that the statement is also true for $k + 1$. To this end, let k be arbitrary but fixed and assume that Feedback Arc Set can be $(1 + \frac{1}{k})$-approximated in $(2 - \delta_k)^n$ time for some $\delta_k > 0$. Let γ be the solution to the equation

$$\frac{\gamma \log(\gamma) - (\gamma - 1)\log(\gamma - 1)}{\gamma - 1} = 1 - \log(2 - \delta_k),$$

where all logarithms use base 2, and let $\alpha = \frac{1}{\gamma}$.[4] Note that $\gamma \geq 2$ exists for each $0 < \delta_k < 1$ as the right side of the equations is some value between zero and one and the left side evaluates to 2 for $\gamma = 2$ and tends towards 0 for increasing values of γ as shown next. Note that the left side of the previous equation is equivalent to $(\log(\gamma) - \log(\gamma - 1)) + \frac{\log(\gamma)}{\gamma - 1}$. The first summand describes the derivative of the logarithm (which is $\frac{1}{\ln(2)\gamma}$) and therefore tends towards 0 for increasing values of γ. The second summand also clearly tends towards zero as the logarithm grows sublinearly.

We next describe the $(1 + \frac{1}{k+1})$-approximation for Feedback Arc Set. Let $G = (V, A)$ be the input graph and let n be the number of vertices in V. For the sake of simplicity, we will assume that αn is an integer to avoid dealing with rounding issues (which will vanish in the Bachmann-Landau notation anyway). Bodlaender et al. [6] showed how to solve Feedback Arc Set exactly in $O^*(2^n)$ time. Importantly, their algorithm iterates over all induced subgraphs (in order of increasing size), and computes the optimal solution for each induced subgraph in polynomial time for each subgraph. We use their algorithm for all induced subgraphs with at most αn vertices. Let $\mathcal{V}$ be the set of all sets of vertices of size exactly αn. Using the algorithm by Bodlaender et al. [6] to compute the optimal solution for all graphs induced by sets in $\mathcal{V}$ takes $O^*(\binom{n}{\alpha n})$ time. In the same time, we can also compute for each $V' \in \mathcal{V}$ the number of arcs in $\{(u, v) \mid u \in V \setminus V' \wedge v \in V'\}$. Let $a(V')$ denote this number and let $V^* \in \mathcal{V}$ be a set minimizing $a(V^*)$. We then use the algorithm by Bodlaender et al. [6] to solve the instance $G[V \setminus V^*]$ optimally in $O^*(2^{(1-\alpha)n})$ time and for all other $V' \in \mathcal{V}$, we use the $(1 + \frac{1}{k})$-approximation for $G[V \setminus V']$ that runs in $O((2 - \delta_k)^{(1-\alpha)n})$ time. Finally for each $V' \in \mathcal{V}$, we compute the sum of the optimal value for $G[V']$ plus $a(V')$ plus the size of the computed solution for $G[V \setminus V']$. We return the smallest number found (or the set associated to the smallest number if we want to return the feedback arc set).

It remains to show that our algorithm always produces a $(1 + \frac{1}{k+1})$-approximation and to analyze the running time. For the approximation factor, let F be a feedback arc set of G of minimum size and let $\mathrm{OPT} = |F|$. We distinguish between the following two cases. Either $a(V^*) \leq \frac{1}{k+1}\,\mathrm{OPT}$ or $a(V^*) > \frac{1}{k+1}\,\mathrm{OPT}$. In the first case, note that the solution we computed for V^* consists of optimal solutions for $G[V^*]$ and $G[V \setminus V^*]$ plus $a(V^*)$. Since the first two summands are combined at most OPT and $a(V^*) \leq \frac{1}{k+1}\,\mathrm{OPT}$, we indeed return a feedback arc set of size at most $(1 + \frac{1}{k+1})\,\mathrm{OPT}$. If $a(V^*) > \frac{1}{k+1}\,\mathrm{OPT}$, then note that for each $V' \in \mathcal{V}$ it holds that $a(V') > \frac{1}{k+1}\,\mathrm{OPT}$. Moreover, since the graph $G' = (V, A \setminus F)$ is acyclic, it has a topological ordering. Let V'' be the set of the first αn vertices in that topological ordering. Let d be the size of a minimum feedback arc set in $G[V \setminus V'']$ and let $c = \frac{d}{\mathrm{OPT}}$. Note that $c \leq \frac{k}{k+1}$. Finally, the solution we compute for V'' has size

[4] We note that γ is roughly $-\frac{W(-\frac{(2-\delta_k)(1-\log(2-\delta_k))}{2})}{1-\log(2-\delta_k)}$, where W is the Lambert W function (also known as the product logarithm). It is known that this function cannot be expressed in terms of elementary functions.

$$(1-c)\,\mathrm{OPT}+(1+\frac{1}{k})c\,\mathrm{OPT} \leq \mathrm{OPT}-c\,\mathrm{OPT}+c\,\mathrm{OPT}+\frac{1}{k}\frac{k}{k+1}\,\mathrm{OPT} = (1+\frac{1}{k+1})\,\mathrm{OPT}\,.$$

Thus, we also return a $(1+\frac{1}{k+1})$-approximation in this case.

Let us now analyze the running time. Recall that $\gamma = \frac{1}{\alpha}$. Computing $\mathcal{V}$, $a(V')$, and the optimal solution for $G[V']$ for each $V' \in \mathcal{V}$ take $O^*(\binom{n}{\alpha n})$ time. Solving the instance $G[V \setminus V^*]$ optimally takes $O^*(2^{(1-\alpha)n})$ time. Computing the approximate solutions for all sets in $\mathcal{V}$ takes overall $O^*(\binom{n}{\alpha n} \cdot (2-\delta_k)^{(1-\alpha)n})$ time. By standard arguments, $\binom{n}{\alpha n} \in O((\frac{\gamma^\gamma}{(\gamma-1)^{\gamma-1}})^{\alpha n})$ [35]. Hence

$$\begin{aligned}
O(\binom{n}{\alpha n} \cdot (2-\delta_k)^{(1-\alpha)n}) &= O(2^{\log(\frac{\gamma^\gamma}{(\gamma-1)^{\gamma-1}})^{\alpha n})} \cdot 2^{\log((2-\delta_k)^{(1-\alpha)n})}) \\
&\subseteq O(2^{\alpha n(\log(\gamma^\gamma)-\log((\gamma-1)^{\gamma-1}))+(1-\alpha)n\log(2-\delta_k)}) \\
&\subseteq O(2^{(1-\alpha)n\left(\frac{\alpha}{1-\alpha}(\gamma\log(\gamma)-(\gamma-1)\log(\gamma-1))+\log(2-\delta_k)\right)}) \\
&\subseteq O(2^{(1-\alpha)n\left(\frac{\gamma\log(\gamma)-(\gamma-1)\log(\gamma-1)}{\gamma-1}+\log(2-\delta_k)\right)}) \\
&\subseteq O(2^{(1-\alpha)n}),
\end{aligned}$$

where again all logarithms use base 2. The last equality is due to the definition of γ. Thus, the total running time of our algorithm is $O^*(2^{(1-\alpha)n})$. This is in $O((2-\delta_{k+1})^n)$ time for any $0 < \delta_{k+1} < 2-2^{1-\alpha}$. This completes the induction step and concludes the proof. ◀

3.2 Cutwidth

We continue with Directed Cutwidth. We again show how to use Proposition 1 to compute a 2-approximaiton.

▶ **Proposition 4.** *Directed Cutwidth can be 2-approximated in $O^*(2^{\frac{\omega n}{3}})$ time.*

Proof. Let $G = (V, A)$ be an input to Directed Cutwidth. We first find an optimal solution L to the instance $(G, \lfloor \frac{n}{2} \rfloor)$ of Directed Minimum $(k, n-k)$-Cut in $O^*(2^{\frac{\omega n}{3}})$ time using Proposition 1. Let $R = V \setminus L$ and let A^* be the set of arcs (u, v) with $u \in R$ and $v \in L$. Note that the size of A^* is by definition a lower bound for the directed cutwidth of G. Next, we can solve $G[L]$ and $G[R]$ optimally in $O^*(2^{\frac{n}{2}})$ time [23]. Since adding a vertex and/or arc to a graph cannot decrease its directed cutwidth, we have that the cutwidth of $G[L]$ and $G[R]$ are both at most the cutwidth of G. Hence, placing all vertices in L according to the computed solution for $G[L]$ and then all vertices in R according to the solution for $G[R]$ gives an ordering for the graph $G' = (V, A \setminus A^*)$ that shows that the cutwidth of G' is at most the cutwidth of G. See Figure 3 for an example. Adding the set A^* to G' can increase the cutwidth by at most $|A^*|$ and adding arcs (u, v) with $u \in L$ and $v \in R$ does not increase the directed cutwidth. Thus, the computed ordering is a 2-approximation as the size of A^* is at most the cutwidth of G as shown above. Note that the running time is in $O^*(2^{\frac{\omega n}{3}} + 2^{\frac{n}{2}}) = O^*(2^{\frac{\omega n}{3}})$. This concludes the proof. ◀

Unfortunately, due to the inherent maxima nature of Directed Cutwidth, we do not know how to apply a boosting technique similar to Theorem 3 to get an approximation factor smaller than 2.

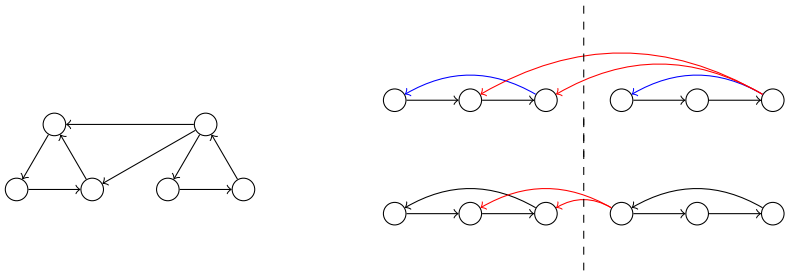

Figure 3 An example of Directed Cutwidth. The input instance is depicted on the left with the exception that all vertices in the left triangle have arcs towards all vertices in the right triangle unless the respective reverse arc is present. We decided to not show these arcs for the sake of readibility. On the right, two optimal solutions to Directed Minimum $(k, n-k)$-Cut with $k = 3$ are shown (red arcs) with optimal orderings for the two respective subproblems of Directed Cutwidth are depicted. The ordering shown on the top has a directed cutwidth of 3 (e.g. at positions 4 and 5) and the ordering on the bottom has a directed cutwidth of 2.

3.3 Optimal Linear Arrangement

In this section, we study Optimal Linear Arrangement. We show how to compute a $(2+\varepsilon)$-approximation for Directed Optimal Linear Arrangement for each $\varepsilon > 0$ in $O((2-\delta_\varepsilon)^n)$ time for some $\delta_\varepsilon > 0$ only depending on ε. We then show how to improve the algorithm in the undirected setting to achieve a $(\frac{3}{2}+\varepsilon)$-approximation in the same time.

▶ **Proposition 5.** *For every $0 < \alpha < 1$, there is a $(1 + \frac{1}{(1-\alpha)})$-approximation for Directed Optimal Linear Arrangement that runs in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{2})n})$ time, where ω is the matrix-multiplication exponent.*

Proof. Let $G = (V, A)$ be an input to Directed Optimal Linear Arrangement. We use Proposition 1 to solve Directed Minimum $(k, n-k)$-Cut for each $\frac{\alpha n}{2} \leq k \leq n - \frac{\alpha n}{2}$. This takes $O^*(2^{\frac{\omega n}{3}})$ time. Let k' be a value where the number of arcs from $V \setminus L$ to L are minimum for some set L of size k'. Let L' be a set of vertices of size k' minimizing the number of arcs from $R' = V \setminus L'$ to L' and let A' be the set of arcs from R' to L'. By the pigeonhole principle, the size of A' is at most $\frac{\mathrm{OPT}}{(1-\alpha)n}$ where OPT is the optimal directed-optimal-linear-arrangement value for G (the value we want to minimize in Directed Optimal Linear Arrangement). Note that $|L'|, |R'| \leq (1 - \frac{\alpha}{2})n$. Next, we solve the instances $G[L]$ and $G[R]$ optimally in $O^*(2^{(1-\frac{\alpha}{2})n})$ time using the algorithm by Bodlaender et al. [6]. To get an approximation for the whole graph G, we first order all vertices in L according to the computed solution for $G[L]$ and then all vertices in R according to the computed solution for R. Note that this is an optimal solution for the graph $G' = (V, A \setminus A')$. It remains to add the arcs between L and R. Note that each edge in A' can contribute at most n and therefore the edges in A' can contribute at most $\frac{\mathrm{OPT}}{1-\alpha}$ in total. Arcs from L to R do not contribute as they are forward arcs in the computed ordering. Added to the cost for G' (which is at most OPT), this yields the claimed approximation factor. The running time is in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{2})n})$. ◀

We mention that the above result yields a 2.72-approximation in $O^*(2^{\frac{\omega n}{3}})$ time, assuming the currently known value of $\omega < 2.372$. We next show how to improve the approximation factor to $(\frac{3}{2}+\varepsilon)$ in the undirected setting. The general strategy is the same but in the end, we use the fact that an ordering and its reverse have the same value in terms of Optimal Linear Arrangement and leverage this fact to argue that contribution of each edge between the two parts is at most $\frac{n}{2}$ on average.

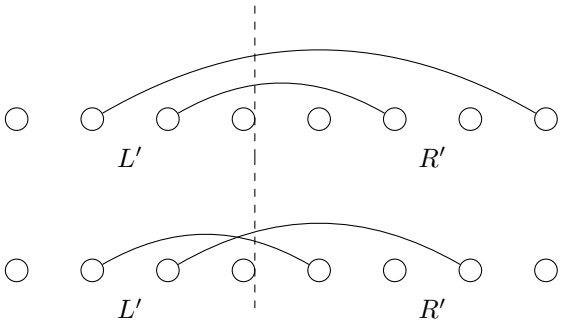

Figure 4 An example of Optimal Linear Arrangement where the vertices are divided into two parts L' and R' with two edges between the two parts and some ordering within the two parts is fixed. For the sake of simplicity, we do not show edges within the two parts. The two orderings only differ in that we reverse the ordering of the vertices in R'. The average length within L' of the two edges between L' and R' is 1.5 in both orderings. The average length within R' is 3 in the above ordering and $2'$ in the below ordering.

▶ **Proposition 6.** *For each constant $0 < \alpha < 1$, there is a $(1 + \frac{1}{2(1-\alpha)})$-approximation for Optimal Linear Arrangement that runs in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{2})n})$ time, where ω is the matrix-multiplication exponent.*

Proof. Let $G = (V, E)$ be an input graph for Optimal Linear Arrangement. We start the same as in Proposition 5, that is, we solve Minimum $(k, n-k)$-Cut for each $\frac{\alpha n}{2} \leq k \leq \frac{n}{2}$, define sets L', R', and E' and value OPT, and solve $G[L']$ and $G[R']$ optimally in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{2})n})$ time overall. Let $i = |L'|$. We next test for both L' and R' whether the average length of edges in E' is smaller if we order the vertices according to the computed solution or in the reverse order. To this end, we pretend that the endpoint outside the respective part is fixed at position i. See Figure 4 for an example. We pick for each of the two the ordering which results in shorter average lengths and observe that the average length of an edge in E' is now at most $\frac{n}{2}$ as the average length "in L'" is at most $\frac{|L'|-1}{2}$, the average length "in R'" is at most $\frac{|R'|+1}{2}$, and $|L'| + |R'| = n$. The combined cost for all edges in E' is therefore at most $\frac{\text{OPT}}{(1-\alpha)n} \cdot \frac{n}{2} = \frac{\text{OPT}}{2(1-\alpha)}$ and added to the cost for G' (which is at most OPT), this yields the claimed approximation factor. The running time is in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{2})n})$. ◀

The above results yields a 1.86-approximation in $O^*(2^{\frac{\omega n}{3}})$ time using the currently known value of $\omega \approx 2.372$.

3.4 Pathwidth

We conclude this section with a 2-approximation for Directed Pathwidth in $O(1.66^n)$ time. We point out that this result stands out from the previous in two ways: Directed Pathwidth can be solved in $O(1.89^n)$ time and our algorithm is not based on Proposition 1.

▶ **Theorem 7.** *Directed Pathwidth can be 2-approximated in $O(1.66^n)$ time.*

Proof. Let $G = (V, A)$ be the input graph for which we want to approximate the pathwidth, let $\alpha \approx 0.204$ be the solution to the equation $1.89^{1-\alpha} = (\frac{1}{\alpha})^{\alpha}(\frac{1}{1-\alpha})^{1-\alpha}$, and let OPT be the (unknown) directed pathwidth of G. We first use the algorithm by Bodlaender et al. [6] to find a set V' of αn vertices (and an ordering of these vertices) such that independent of the ordering of the remaining $(1-\alpha)n$ vertices, it holds for each position i that at most OPT

vertices before position $\min(i, \alpha n)$ have in-neighbors after position i. This can be done in time $O^*((\frac{1}{\alpha})^{\alpha n}(\frac{1}{1-\alpha})^{(1-\alpha)n}) = O^*(1.89^{(1-\alpha)n})$. Next, we solve the instance $G[V \setminus V']$ of Directed Pathwidth optimally in $O^*(1.89^{(1-\alpha)n})$ time [30]. We return the ordering that places the vertices in V' first (in the order that was computed before) and then all vertices in $V \setminus V'$ in the order corresponding to the solution for $G[V \setminus V']$. Note that it holds for each position $i > \alpha n$ that at most OPT vertices before position αn have in-neighbors after position i and at most OPT vertices between position $\alpha n + 1$ and i have in-neighbors after position i, that is, the computed ordering is a 2-approximation. The running time is in $O^*(1.89^{(1-\alpha)n}) \approx O^*(1.89^{(0.796)n}) \subseteq O(1.66^n)$. This concludes the proof. ◀

4 Weighted Problems

In this section, we consider weighted generalizations of the problems studied in the last section. We denote the largest weight in the input by $w_{\max}$. We mention that if $w_{\max}$ is polynomially upper-bounded in n, then all of our previous results generalize in a straight-forward way. Hence, we mainly focus on cases where $w_{\max}$ is large compared to n. An assumption that is often made when dealing with weighted problems is that all numbers can be added and subtracted in constant time. However, in the realm of very large weights, this assumption is highly unrealistic as even reading the weight from input takes $O(\log(w_{\max}))$ time (which we sometimes hide in the O^*-notation). We decided to not make the above assumption which basically adds an additional $O(\log(w_{\max}))$ factor to all running times.

We mention that we do not study a weighted version of Directed Pathwidth. To the best of our knowledge, this problem has only been studied once [32] and our algorithm behind Theorem 7 computes a 2-approximation even in the weighted setting if we have an exact (slower) algorithm. We suspect that the $O(1.89^n)$-time algorithm by Kitsunai et al. [30] generalizes to the weighted setting but this was never proven and proving it here would go beyond the scope of this paper.

We start with an arc-weighted generalization of Directed Minimum $(k, n-k)$-Cut. We mention that if this generalization can be solved in $O^*(2^{\frac{\omega n}{3}})$ time, then our previous algorithms for Feedback Arc Set, Directed Optimal Linear Arrangement, and Directed Cutwidth can all easily be generalized to the weighted setting. Unfortunately, even for (undirected) Weighted Max Cut, no such algorithm is known. However, Williams [36] showed how to compute a $(1+\varepsilon)$-approximation for Weighted Max Cut in $O^*(2^{\frac{\omega n}{3}}(\frac{1}{\varepsilon})^3)$ time. We will generalize this result to Weighted Directed Minimum $(k, n-k)$-Cut and show how to use this approximation for the other problems.

▶ **Proposition 8.** *For any $\varepsilon > 0$, Weighted Directed Minimum $(k, n-k)$-Cut can be $(1+\varepsilon)$-approximated in $O(2^{\frac{\omega n}{3}} \cdot \log(w_{\max}) \cdot (\frac{1}{\varepsilon})^2 \cdot k^2)$ time.*

Proof. Let $(G = (V, A), k)$ be an input instance and let $\varepsilon > 0$ be an arbitrary but fixed value. We partition V arbitrarily into three sets V_1, V_2, and V_3 of (roughly) equal size. Next, we guess in $O(k^2)$ time how many vertices of V_1, V_2, and V_3 belong to an optimal solution A. Let these numbers be k_1, k_2 and $k_3 = k - k_1 - k_2$. Next, we build an undirected graph H with edge weights as follows. For each $i \in [3]$, we introduce a vertex v_j^i for each subset $V_j^i \subseteq V_i$ with $|V_j^i| = k_i$. Note that the number of vertices in H is in $O(2^{n/3})$. For each vertex v_j^i, let $d(v_j^i)$ denote the sum of weights of all arcs from vertices in V_j^i to vertices in $V_i \setminus V_j^i$ in G. For each pair of vertices $v_j^i, v_{j'}^{i'}$ with $i \neq i'$, we add an edge between them of weight

$$\frac{d(v_j^i) + d(v_{j'}^{i'})}{2} + \sum_{\{(u,w) \in A \mid u \in V_j^i \wedge w \in V_{i'} \setminus V_{j'}^{i'}\} \cup \{(u,w) \in A \mid u \in V_{j'}^{i'} \wedge w \in V_i \setminus V_j^i\}} w((u,w)).$$

Using an algorithm by Zwick [38] (see also Williams [36]), we find a $(1+\varepsilon)$-approximation for the minimum-weight triangle in H in $O(2^{\frac{\omega n}{3}} \frac{1}{\varepsilon} \log(\frac{n^2 w_{\max}}{\varepsilon})) \subseteq O(2^{\frac{\omega n}{3}} (\frac{1}{\varepsilon})^2 \log(w_{\max}))$ time. Note that $\log(n^2 w_{\max}) \leq \log({w_{\max}}^3) \leq 3\log(w_{\max})$ as we assume $w_{\max} > n$. Moreover, each triangle in H corresponds to a directed $(k, n-k)$-cut in H of the same weight and each directed $(k, n-k)$-cut with k_i vertices in V_i for each $i \in [3]$ corresponds to a triangle of the same weight in H. Hence, we compute a $(1+\varepsilon)$-approximation for Weighted Directed Minimum $(k, n-k)$-Cut. The total running time is in $O(2^{\frac{\omega n}{3}} \log(w_{\max})(\frac{1}{\varepsilon})^2 k^2)$ and this concludes the proof. ◀

4.1 Feedback Arc Set

We next show how to use Proposition 8 to get faster approximation algorithms for the weighted variants of the other vertex-ordering problems we consider. We start with Weighted Feedback Arc Set. Similar to the unweighted case, we first show a constant factor approximation and then show a boosting technique to achieve any $(1+\varepsilon)$-approximation.

▶ **Proposition 9.** *Weighted Feedback Arc Set admits a 3-approximation that can be computed in $O(2^{\frac{\omega n}{3}} \log(w_{\max})k^2)$ time.*

Proof. Let $G = (V, A)$ be an input graph for Weighted Feedback Arc Set. We first compute a 2-approximation L for the instance $(G, \lfloor \frac{n}{2} \rfloor)$ of Weighted Directed Minimum $(k, n-k)$-Cut in $O(2^{\frac{\omega n}{3}} \log(w_{\max})k^2)$ time using Proposition 8 with $\varepsilon = 1$. Note that half the sum of weights of arcs from $R = V \setminus L$ to L is a lower bound for the size of a minimum feedback arc set. We refer to Figure 2 for an illustration and mention that this time the weight of all red arcs is a 2-approximation for an optimal cut. Next, we solve the two instances $G[L]$ and $G[R]$ of Weighted Feedback Arc Set optimally in $O(2^{\frac{n}{2}} n^2 \log(w_{\max}))$ time [23]. Placing the vertices of L in the computed order before the vertices of R in the computed order yields a 3-approximation as the weights of the feedback arcs within both instances are combined also a lower bound for the weight of a minimum-weight feedback arc set in G. The running time is in $O(2^{\frac{\omega n}{3}} \log(w_{\max})k^2)$ as $\frac{\omega}{3} > \frac{1}{2}$. ◀

We next present the boosting technique in the weighted setting. It is very similar to the unweighted setting with two minor adjustments: the formulas change slightly due to the fact that we start with a 3-approximation rather than a 2-approximation and an additional factor of $O(\log(w_{\max}))$ is used when using the algorithm by Bodlaender et al. [6]. We present the entire proof for the sake of completeness.

▶ **Theorem 10.** *For each $\varepsilon > 0$, there is a $\delta > 0$ such that Weighted Feedback Arc Set can be $(1+\varepsilon)$-approximated in $O((2-\delta)^n \cdot \log(w_{\max}))$ time.*

Proof. We prove this statement by showing via induction that for each $k \geq 1$, there is a $\delta > 0$ such that Weighted Feedback Arc Set can be $(1 + \frac{2}{k})$-approximated in $O((2-\delta)^n \log(w_{\max}))$ time. Since any $\varepsilon > 0$ is larger than $\frac{2}{k}$ for some value of k, this will prove the theorem. Note that the base case $k = 1$ is proven by Proposition 9. It remains to show the induction step, that is, assuming that the statement is true for some value of k, we need to show that the statement is also true for $k+1$. To this end, let k be arbitrary but fixed and assume that Weighted Feedback Arc Set can be $(1 + \frac{2}{k})$-approximated in $(2-\delta_k)^n$ time for some $\delta_k > 0$. Let γ be the solution to the equation

$$\frac{\gamma \log(\gamma) - (\gamma - 1)\log(\gamma - 1)}{\gamma - 1} = 1 - \log(2 - \delta_k),$$

where all logarithms use base 2, and let $\alpha = \frac{1}{\gamma}$.

We next describe how to get the $(1+\frac{2}{k+1})$-approximation for Weighted Feedback Arc Set. Let $G=(V,A)$ be the input graph and let n be the number of vertices in V. For the sake of simplicity, we will again assume that αn is an integer. Let $\mathcal{V}$ be the set of all vertices with exactly αn vertices. Using the algorithm by Bodlaender et al. [6], we compute the optimal solution for all graphs induced by sets in $\mathcal{V}$. This takes $O(\binom{n}{\alpha n}\operatorname{poly}(n)\log(w_{\max}))$ time. In the same time, we can also compute for each $V'\in\mathcal{V}$ the sum of weights of arcs in $\{(u,v)\mid u\in V\setminus V'\wedge v\in V'\}$. Let $a(V')$ denote this number and let $V^*\in\mathcal{V}$ be a set minimizing $a(V^*)$. We then use the algorithm by Bodlaender et al. [6] to solve the instance $G[V\setminus V^*]$ optimally in $O(2^{(1-\alpha)n}\operatorname{poly}(n)\log(w_{\max}))$ time and for all other $V'\in\mathcal{V}$, we use the $(1+\frac{2}{k})$-approximation for $G[V\setminus V']$ that runs in $O((2-\delta_k)^{(1-\alpha)n})$ time. Finally, we for each $V'\in\mathcal{V}$, we compute the sum of the optimal value for $G[V']$ plus $a(V')$ plus the size of the computed solution for $G[V\setminus V']$. We return the smallest number found.

It remains to show that our algorithm always produces a $(1+\frac{2}{k+1})$-approximation and to analyze the running time. For the approximation factor, let F be a feedback arc set of G of minimum weight OPT. We distinguish between the following two cases. Either $a(V^*)\leq\frac{2}{k+1}\operatorname{OPT}$ or $a(V^*)>\frac{2}{k+1}\operatorname{OPT}$. In the first case, note that the solution we computed for V^* consists of optimal solutions for $G[V^*]$ and $G[V\setminus V^*]$ plus $a(V^*)$. Since the first two summands are combined at most OPT and $a(V^*)\leq\frac{2}{k+1}\operatorname{OPT}$, we indeed return a feedback arc set of size at most $(1+\frac{2}{k+1})\operatorname{OPT}$. If $a(V^*)>\frac{2}{k+1}\operatorname{OPT}$, then note that for each $V'\in\mathcal{V}$ it holds that $a(V')>\frac{2}{k+1}\operatorname{OPT}$. Moreover, since the graph $G'=(V,A\setminus F)$ is acyclic, it has a topological ordering. Let V'' be the set of the first αn vertices in that topological ordering. Let d be the weight of a feedback arc set of minimum weight in $G[V\setminus V'']$. Note that $d\leq\frac{k-1}{k+1}\operatorname{OPT}$. The solution we compute for V'' has size $(\operatorname{OPT}-d)+(1+\frac{2}{k})d\leq(1+\frac{2(k-1)}{k(k+1)})\operatorname{OPT}<(1+\frac{2}{k+1})\operatorname{OPT}$. Thus also in this case, we return a $(1+\frac{2}{k+1})$-approximation.

We conclude with analyzing the running time. Recall that $\gamma=\frac{1}{\alpha}$. Computing $\mathcal{V}$, $a(V')$, and the optimal solution for $G[V']$ for each $V'\in\mathcal{V}$ takes $O(\binom{n}{\alpha n}\operatorname{poly}(n)\log(w_{\max}))$ time. Solving the instance $G[V\setminus V^*]$ optimally takes $O(2^{(1-\alpha)n}\operatorname{poly}(n)\log(w_{\max}))$ time. Approximating the solutions for all sets in $\mathcal{V}$ takes $O(\binom{n}{\alpha n}\cdot(2-\delta_k)^{(1-\alpha)n}\cdot\log(w_{\max}))$ time in total. As shown in the proof for Theorem 3, it holds that $O(\binom{n}{\alpha n}\cdot(2-\delta_k)^{(1-\alpha)n})\subseteq O(2^{(1-\alpha)n})$. Thus, the total running time of our algorithm is in $O^*(2^{(1-\alpha)n})$ and in $O((2-\delta)^n\log(w_{\max}))$ for any $0<\delta<2-2^{1-\alpha}$. This concludes the proof. ◀

4.2 Cutwidth and Optimal Linear Arrangement

In this final part, we generalize our algorithms for Directed Cutwidth and Directed Optimal Linear Arrangement to the weighted setting. Since all generalizations follow the same simple structure, we only show the result for Directed Cutwidth.

▶ **Proposition 11.** *Weighted Directed Cutwidth admits a $(2+\varepsilon)$-approximation in time $O(2^{\frac{\omega n}{3}}\log(w_{\max})(\frac{1}{\varepsilon})^2k^2)$.*

Proof. The proof is similar to the proof of Proposition 4. Let $(G=(V,A),w)$ be the input instance to Weighted Directed Cutwidth. We find a $(1+\varepsilon)$-approximation for the instance $(G,w,\lfloor\frac{n}{2}\rfloor)$ of Weighted Directed Minimum $(k,n-k)$-Cut using Proposition 8. This takes $O(2^{\frac{\omega n}{3}}\log(w_{\max})(\frac{1}{\varepsilon})^2k^2)$ time. Let L be the set of vertices returned by Proposition 1, let $R=V\setminus L$, and let $A'\subseteq A$ be the set of arcs from R to L. Note that $\frac{w(A')}{1+\varepsilon}$ is by definition a lower bound for the weighted directed cutwidth of G. Next, we can solve $G[L]$ and $G[R]$ optimally in $O^*(2^{\frac{n}{2}}\log(w_{\max}))$ time [23]. Since adding a vertex

and/or arc to a graph cannot decrease its cutwidth, we have that the weighted directed cutwidth of $G[L]$ and $G[R]$ are both at most the weighted directed cutwidth of G. Adding the edges from L to R also does not increase the cutwidth. Hence, placing all vertices in L according to the computing solution for $G[L]$ and then all vertices in R according to the solution for $G[R]$ gives an ordering for the graph $G' = (V, A \setminus A')$ that shows that the weighted directed cutwidth of G' is at most the weighted directed cutwidth of G. Adding the set A' to G' can increase the cutwidth by at most $w(A')$ and hence the computed ordering is a $(2+\varepsilon)$-approximation. The running time is in $O(2^{\frac{\omega n}{3}} \log(w_{\max})(\frac{1}{\varepsilon})^2 k^2)$ as $\frac{\omega}{3} > \frac{1}{2}$ and this concludes the proof. ◀

The generalizations for Optimal Linear Arrangement and Directed Optimal Linear Arrangement are basically the same and are hence omitted. We just mention that the slightly weaker running time $(1 - \frac{\alpha}{4})$ instead of $(1 - \frac{\alpha}{2})$ in the exponent comes from the fact that we only have a $(1+\varepsilon)$-approximation for Weighted Directed Minimum $(k, n-k)$-Cut (where we set $\varepsilon = \frac{\alpha}{2}$ and then compute the approximation for all $\frac{\alpha}{4}n \leq k \leq (1 - \frac{\alpha}{4})n$ in order to compensate. The correctness then follows from the fact that $\frac{1+\frac{\alpha}{2}}{1-\frac{\alpha}{2}} \leq \frac{1}{1-\alpha}$ for all $\alpha > 0$.

▶ **Proposition 12.** *For each constant $0 < \alpha < 1$, there is a $(1 + \frac{1}{1-\alpha})$-approximation for Directed Optimal Linear Arrangement that runs in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{4})n})$ time.*

▶ **Proposition 13.** *For each constant $0 < \alpha < 1$, there is a $(1 + \frac{1}{2(1-\alpha)})$-approximation for Optimal Linear Arrangement that runs in $O^*(2^{\frac{\omega n}{3}} + 2^{(1-\frac{\alpha}{4})n})$ time.*

5 Conclusion

In this work, we initiated the study of faster exponential-time approximation algorithms for vertex-ordering problems. For Feedback Arc Set we designed an approximation scheme using a novel self-improving technique. For other problems, we showed how to compute a constant-factor approximation. We mention that the best known polynomial-time algorithms do not achieve a factor better than $O(\sqrt{\log(n)})$.

We conclude with a few open problems. One obvious direction is to design a faster exponential-time algorithm for Directed Minimum $(k, n-k)$-Cut– exact or approximate – which would immediately yield faster approximation algorithms for almost all of the problems considered in this paper. Next, there are a lot more (vertex-ordering) problems that are also interesting to study. Examples include Treewidth, Subset Feedback Vertex/Arc Set (here also the restriction to tournament graphs is often studied), Travelling Sales Person, Bandwidth, or Minimum Fill-In. Second, can our constant-factor approximations be improved to $(1+\varepsilon)$ as in the case of Feedback Arc Set? On a related note, we are not aware of any tool to show lower bounds for exponential-time approximations and most known hypotheses like SETH seem to weak to base such lower bounds on. In particular, a reasonably sounding candidate one could call Gap-SETH (a combination of SETH and Gap-ETH) is not true as shown by Alman et al. [1]. Hence, we ask the following question:

> What would be a sensible hypothesis to exclude $(1+\varepsilon)$-approximations in $O((2-\delta_\varepsilon)^n)$ time for some $\delta_\varepsilon > 0$ depending only on ε for different problems?

Finally, all our algorithms use exponential space (here, the bottleneck is our exact algorithm for Directed Minimum $(k, n-k)$-Cut, which first builds an exponential-size graph). Can this be reduced to polynomial space while still beating the fastest known exact algorithms?

References

1. Josh Alman, Timothy M. Chan, and R. Ryan Williams. Faster deterministic and Las Vegas algorithms for offline approximate nearest neighbors in high dimensions. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 637–649. SIAM, 2020. doi:10.1137/1.9781611975994.39.
2. Andris Ambainis, Kaspars Balodis, Janis Iraids, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1783–1793. SIAM, 2019. doi:10.1137/1.9781611975482.107.
3. Per Austrin, Toniann Pitassi, and Yu Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In *Proceedings of the 15th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, pages 13–24. Springer, 2012. doi:10.1007/978-3-642-32512-0_2.
4. Nikhil Bansal, Parinya Chalermsook, Bundit Laekhanukit, Danupon Nanongkai, and Jesper Nederlof. New tools and connections for exponential-time approximation. *Algorithmica*, 81(10):3993–4009, 2019. doi:10.1007/S00453-018-0512-8.
5. Nikhil Bansal, Dor Katzelnick, and Roy Schwartz. On approximating cutwidth and pathwidth. In *Procceedings of the 65th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2024. Accepted for publication. doi:10.1109/FOCS61266.2024.00051.
6. Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. A note on exact algorithms for vertex ordering problems on graphs. *Theory of Computing Systems*, 50(3):420–432, 2012. doi:10.1007/S00224-011-9312-0.
7. Edouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Multi-parameter analysis for local graph partitioning problems: Using greediness for parameterization. *Algorithmica*, 71(3):566–580, 2015. doi:10.1007/S00453-014-9920-6.
8. Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discrete Applied Mathematics*, 159(17):1954–1970, 2011. doi:10.1016/J.DAM.2011.07.009.
9. Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 370–379. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.47.
10. Moses Charikar, Mohammad Taghi Hajiaghayi, Howard J. Karloff, and Satish Rao. ℓ_2^2 spreading metrics for vertex ordering problems. *Algorithmica*, 56(4):577–604, 2010.
11. Johanne Cohen, Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, and Gregory Kucherov. Optimal linear arrangement of interval graphs. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 267–279. Springer, 2006. doi:10.1007/11821069_24.
12. Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On cutwidth parameterized by vertex cover. *Algorithmica*, 68(4):940–953, 2014. doi:10.1007/S00453-012-9707-6.
13. Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theoretical Computer Science*, 411(40-42):3701–3713, 2010. doi:10.1016/J.TCS.2010.06.018.
14. Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.
15. Baris Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Faster exponential-time approximation algorithms using approximate monotone local search. In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA)*, pages 50:1–50:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.50.
16. Baris Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Approximate monotone local search for weighted problems. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 17:1–17:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.17.

17 Baris Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Optimally repurposing existing algorithms to obtain exponential-time approximations. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 314–345. SIAM, 2024. doi:10.1137/1.9781611977912.13.

18 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. doi:10.1007/PL00009191.

19 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. doi:10.1137/05064299X.

20 Uriel Feige and James R. Lee. An improved approximation ratio for the minimum linear arrangement problem. *Information Processing Letters*, 101(1):26–29, 2007. doi:10.1016/J.IPL.2006.07.009.

21 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 764–775. ACM, 2016. doi:10.1145/2897518.2897551.

22 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010. doi:10.1007/978-3-642-16533-7.

23 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 16th ACM national meeting*, page 71. ACM, 1961. doi:10.1145/800029.808532.

24 Thore Husfeldt, Dieter Kratsch, Ramamohan Paturi, and Gregory B. Sorkin, editors. *Exact Complexity of NP-hard Problems*, volume 10441 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2010.

25 Thore Husfeldt, Ramamohan Paturi, Gregory B. Sorkin, and Ryan Williams, editors. *Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time*, volume 13331 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2013.

26 Tanmay Inamdar, Madhumita Kundu, Pekka Parviainen, M. S. Ramanujan, and Saket Saurabh. Exponential-time approximation schemes via compression. In *Proceedings of the 15th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 64:1–64:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.64.

27 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. doi:10.1137/0207033.

28 Pallavi Jain, Lawqueen Kanesh, William Lochet, Saket Saurabh, and Roohani Sharma. Exact and approximate digraph bandwidth. In *Proceedings of the 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 18:1–18:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSTTCS.2019.18.

29 Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.

30 Kenta Kitsunai, Yasuaki Kobayashi, Keita Komuro, Hisao Tamaki, and Toshihiro Tano. Computing directed pathwidth in $O(1.89^n)$ time. *Algorithmica*, 75(1):138–157, 2016.

31 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. Breaking the all subsets barrier for min k-cut. In *Procceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 90:1–90:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.90.

32 Rodica Mihai and Ioan Todinca. Pathwidth is NP-hard for weighted trees. In *Proceedings of the 3rd International Workshop on Frontiers in Algorithmics (FAW)*, pages 181–195. Springer, 2009. doi:10.1007/978-3-642-02270-8_20.

33 Jesper Nederlof. Bipartite TSP in $o(1.9999^n)$ time, assuming quadratic time matrix multiplication. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 40–53. ACM, 2020.

34 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Wegrzycki. A faster exponential time algorithm for bin packing with a constant number of bins via additive combinatorics. *SIAM Journal of Comput.*, 52(6):1369–1412, 2023. doi:10.1137/22M1478112.

35 Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.

36 Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1227–1237. Springer, 2004. doi:10.1007/978-3-540-27836-8_101.

37 Or Zamir. Breaking the 2^n barrier for 5-coloring and 6-coloring. In *Procceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 113:1–113:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

38 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.