The Maximum Binary Tree Problem

Karthekeyan Chandrasekaran
Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
karthe@illinois.edu

Elena Grigorescu
Purdue University, West Lafayette, IN, USA
elena-g@purdue.edu

Gabriel Istrate
West University of Timişoara, Romania
e-Austria Research Institute, Timişoara, Romania
gabrielistrate@acm.org

Shubhang Kulkarni
Purdue University, West Lafayette, IN, USA
kulkar17@purdue.edu

Young-San Lin
Purdue University, West Lafayette, IN, USA
lin532@purdue.edu

Minshen Zhu
Purdue University, West Lafayette, IN, USA
zhu628@purdue.edu

Abstract

We introduce and investigate the approximability of the *maximum binary tree problem* (MBT) in directed and undirected graphs. The goal in MBT is to find a maximum-sized binary tree in a given graph. MBT is a natural variant of the well-studied longest path problem, since both can be viewed as finding a maximum-sized tree of bounded degree in a given graph.

The connection to longest path motivates the study of MBT in directed acyclic graphs (DAGs), since the longest path problem is solvable efficiently in DAGs. In contrast, we show that MBT in DAGs is in fact hard: it has no efficient $\exp(-O(\log n/\log\log n))$-approximation algorithm under the exponential time hypothesis, where n is the number of vertices in the input graph. In undirected graphs, we show that MBT has no efficient $\exp(-O(\log^{0.63} n))$-approximation under the exponential time hypothesis. Our inapproximability results rely on self-improving reductions and structural properties of binary trees. We also show constant-factor inapproximability assuming $\mathbf{P} \neq \mathbf{NP}$.

In addition to inapproximability results, we present algorithmic results along two different flavors: (1) We design a randomized algorithm to verify if a given directed graph on n vertices contains a binary tree of size k in $2^k\mathsf{poly}(n)$ time. (2) Motivated by the longest heapable subsequence problem, introduced by Byers, Heeringa, Mitzenmacher, and Zervas, *ANALCO 2011*, which is equivalent to MBT in *permutation DAGs*, we design efficient algorithms for MBT in bipartite permutation graphs.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases maximum binary tree, heapability, inapproximability, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.30

Related Version `https://arxiv.org/abs/1909.07915`

Funding *Karthekeyan Chandrasekaran*: Supported by NSF CCF-1814613 and NSF CCF-1907937.
Elena Grigorescu: Supported by NSF CCF-1910659 and NSF CCF-1910411.
Gabriel Istrate: Supported by a grant of Ministry of Research and Innovation, CNCS - UEFISCDI project number PN-III-P4-ID-PCE-2016-0842, within PNCDI III.
Young-San Lin: Supported by NSF CCF-1910659 and NSF CCF-1910411.
Minshen Zhu: Supported by NSF CCF-1910659 and NSF CCF-1910411.

1 Introduction

A general degree-constrained subgraph problem asks for an optimal subgraph of a given graph with specified properties while also satisfying degree constraints on all vertices. Degree-constrained subgraph problems have numerous applications in the field of network design and consequently, have been studied extensively in the algorithms and approximation literature [1, 15–17, 29, 31, 32]. In this work, we introduce and study the *maximum binary tree problem* in directed and undirected graphs. In the maximum binary tree problem (MBT), we are given an input graph G and the goal is to find a *binary tree* in G with maximum number of vertices.

Our first motivation for studying MBT arises from the viewpoint that it is a variant of the longest path problem: In the longest path problem, the goal is to find a maximum-sized tree in which every vertex has degree at most 2. In MBT, the goal is to find a maximum-sized tree in which every vertex has degree at most 3. Certainly, one may generalize both these problems to finding a *maximum-sized degree-constrained tree* in a given graph. In this work we focus on binary trees; however, all our results extend to the maximum-sized degree-constrained tree problem for *constant* degree bound.

Our second motivation for studying MBT is its connection to the *longest heapable subsequence problem* introduced by Byers, Heeringa, Mitzenmacher, and Zervas [10]. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ be a permutation on n elements. Byers et al. define a subsequence (not necessarily contiguous) of σ to be *heapable* if the elements of the subsequence can be sequentially inserted to form a binary *min-heap* data structure. Namely, insertions subsequent to the first element, which takes the root position, happen below previously placed elements. The longest heapable subsequence problem asks for a maximum-length heapable subsequence of a given sequence. This generalizes the well-known longest increasing subsequence problem. Porfilio [30] showed that the longest heapable subsequence problem is equivalent to MBT in permutation directed acyclic graphs (abbreviated *permutation DAGs*): a permutation DAG associated with the sequence σ is obtained by introducing a vertex u_i for every sequence element σ_i, and arcs (u_i, u_j) for every pair (i, j) such that $i > j$ and $\sigma_i \geq \sigma_j$. On the other hand, for sequences of intervals the maximum binary problem is easily solvable by a greedy algorithm [6] (see also [21] for further results and open problems on the heapability of partial orders). These results motivate the study of MBT in restricted graph families.

We now formally define MBT in undirected graphs, which we denote as UndirMaxBinaryTree. A *binary tree* of an *undirected* graph G is a subgraph T of G that is connected and acyclic with the degree of u in T being at most 3 for every vertex u in T. In UndirMaxBinaryTree, the input is an undirected graph G and the goal is to find a binary tree in G with maximum number of vertices. In the rooted variant of this problem, the input is an undirected graph G along with a specified root vertex r and the goal is to find a binary tree containing r in G with maximum number of vertices such that the degree of r in the tree is at most 2. We focus on the unrooted variant of the problem and mention that it reduces to the rooted variant. We emphasize that a binary tree T of G is not necessarily spanning (i.e., may not contain all vertices of the given graph). The problem of verifying whether a given undirected graph has a *spanning* binary tree is **NP**-complete. This follows by a reduction from the Hamiltonian path problem: Given an undirected graph $G = (V, E)$, create a pendant vertex v' adjacent to v for every vertex $v \in V$. The resulting graph has a spanning binary tree if and only if G has a Hamiltonian path.

Next, we formally define MBT in directed graphs. A *tree of a directed graph* G is a subgraph T of G such that T is acyclic and has a unique vertex, termed as the root, with the property that every vertex v in T has a unique directed path *to the root* in T. A *binary tree*

of a directed graph G is a tree T such that the incoming-degree of every vertex u in T is at most 2 while the outgoing-degree of every vertex u in T is at most 1. In the rooted variant of the maximum binary tree problem for directed graphs, the input is a directed graph G along with a specified root r and the goal is to find an r-rooted binary tree T in G with maximum number of vertices. The problem of verifying whether a given directed graph has a *spanning* binary tree is **NP**-complete (by a similar reduction as that for undirected graphs).

The connection to the longest path problem as well as the longest heapable subsequence problem motivates the study of the maximum binary tree problem in directed acyclic graphs (DAGs). In contrast to directed graphs, the longest path problem in DAGs can be solved in polynomial-time (e.g., using dynamic programming or LP-based techniques). Moreover, verifying whether a given DAG contains a spanning binary tree is solvable in polynomial-time using the following characterization: a given DAG on vertex set V contains a spanning binary tree if and only if the partition matroid corresponding to the in-degree of every vertex being at most two and the partition matroid corresponding to the out-degree of every vertex being at most one have a common independent set of size $|V| - 1$. These observations raise the intriguing possibility of solving the maximum binary tree problem in DAGs in polynomial-time. For this reason, we focus on DAGs within the family of directed graphs in this work. We denote the maximum binary tree problem in DAGs as DAGMaxBinaryTree.

The rooted and the unrooted variants of the maximum binary tree problem in DAGs are polynomial-time equivalent by simple transformations. Indeed, the unrooted variant can be solved by solving the rooted variant for every choice of the root. To see the other direction, suppose we would like to find a maximum r-rooted binary tree in a given DAG $G = (V, E)$. Then, we discard from G all outgoing arcs from r and all vertices that cannot reach r (i.e., we consider the sub-DAG induced by the descendents of r) and find an unrooted maximum binary tree in the resulting DAG. If this binary tree is rooted at a vertex $r' \neq r$, then it can be extended to an r-rooted binary tree by including an arbitrary $r' \to r$ path P – since the graph is a DAG, any such path P will not visit a vertex that is already in the tree (apart from r'). The equivalence is also approximation preserving. For this reason, we only study the rooted variant of the problem in DAGs.

We present inapproximability results for MBT in DAGs and undirected graphs. On the algorithmic side, we show that MBT in directed graphs is fixed-parameter tractable when parameterized by the solution size. We observe that the equivalence of the longest heapable subsequence to MBT in permutation DAGs motivates the study of MBT even in restricted graph families. As a first step towards understanding MBT in permutation DAGs, we design an algorithm for bipartite permutation graphs. We use a variety of tools including self-improving and gadget reductions for our inapproximability results, and algebraic and structural techniques for our algorithmic results.

1.1 Related work

Degree-constrained subgraph problems appeared as early as 1978 in the textbook of Garey and Johnson [18] and have garnered plenty of attention in the approximation community [1, 15–17, 23, 29, 31, 32]. A rich line of works have addressed the minimum degree spanning tree problem as well as the minimum cost degree-constrained spanning tree problem leading to powerful rounding techniques and a deep understanding of the spanning tree polytope [12, 13, 16, 19, 25, 28, 32]. Approximation and bicriteria approximation algorithms for the counterparts of these problems in directed graphs, namely degree-constrained arborescence and min-cost degree-constrained arborescence, have also been studied in the literature [7].

In the *maximum-edge* degree-constrained connected subgraph problem, the goal is to find a connected degree-constrained subgraph of a given graph with maximum number of edges. This problem does not admit a PTAS [3] and has been studied from the perspective of fixed-parameter tractability [4]. MBT could be viewed as a *maximum-vertex* degree-constrained connected subgraph problem, where the goal is to maximize the number of *vertices* as opposed to the number of *edges* – the degree-constrained connected subgraph maximizing the number of vertices may be assumed to be acyclic and hence, a tree. It is believed that the *connectivity constraint* makes the maximum-edge degree-constrained connected subgraph problem to become extremely difficult to approximate. Our results formalize this belief when the objective is to maximize the number of vertices.

Switching the objective with the constraint in the maximum-vertex degree-constrained connected subgraph problem leads to the minimum-degree k-tree problem: here the goal is to find a minimum degree subgraph that is a tree with at least k vertices. Minimum degree k-tree admits a $O(\sqrt{(k/\Delta^*)\log k})$-approximation, where Δ^* is the optimal degree and does not admit a $o(\log n)$-approximation [23]. We note that the hardness reduction here (from set cover) crucially requires the optimal solution value Δ^* to grow with the number n of vertices in the input instance, and hence, does not imply any hardness result for input instances in which Δ^* is a constant. Moreover, the approximation result implies that a tree of degree $O(\sqrt{k\log k})$ containing k vertices can be found in polynomial time if the input graph contains a constant-degree tree with k vertices.

We consider the maximum binary tree problem to be a generalization of the longest path problem as both can be viewed as asking for a maximum-sized degree-constrained connected acyclic subgraph. The longest path problem in undirected graphs admits an $\Omega\left((\log n/\log\log n)^2/n\right)$-approximation [9], but it is APX-hard and does not admit a $2^{-O\left(\log^{1-\varepsilon} n\right)}$-approximation for any constant $\varepsilon > 0$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\log^{O(1/\varepsilon)} n}\right)$ [22]. Our hardness results for the max binary tree problem in undirected graphs bolsters this connection. The longest path problem in directed graphs is much harder: For every $\varepsilon > 0$ it cannot be approximated to within a factor of $1/n^{1-\varepsilon}$ unless $\mathbf{P} = \mathbf{NP}$, and it cannot be approximated to within a factor of $(\log^{2+\varepsilon} n)/n$ under the Exponential Time Hypothesis [9]. However, the longest path problem in DAGs is solvable in polynomial time. Our hardness results for the max binary tree problem in DAGs are in stark contrast to the polynomial-time solvability of the longest path problem in DAGs.

On the algorithmic side, the color-coding technique introduced by Alon, Yuster, and Zwick [2] can be used to decide whether an undirected graph $G = (V, E)$ contains a copy of a bounded treewidth pattern graph $H = (V_H, E_H)$ where $|V_H| = O(\log |V|)$, and if so, then find one in polynomial time. The idea here is to randomly color the vertices of G by $O(\log |V|)$ colors and to find a maximum colorful copy of H using dynamic programming. We note that the same dynamic programming approach can be modified to find a maximum colorful binary tree. This algorithm can be derandomized, thus leading to a deterministic $\Omega((1/n)\log n)$-approximation to UndirMaxBinaryTree.

In parameterized complexity, designing algorithms with running time $\beta^k\mathsf{poly}(n)$ ($\beta > 1$ is a constant) for problems like k-Path and k-Tree is a central topic. For k-Path, the color-coding technique mentioned above already implies a $(2e)^k\mathsf{poly}(n)$-time algorithm. Koutis [26] noticed that k-Path can be reduced to detecting whether a given polynomial contains a multilinear term. Using algebraic methods for the latter problem, Koutis obtained a $2^{1.5k}\mathsf{poly}(n)$ time algorithm for k-Path. This was later improved by Williams [36] to $2^k\mathsf{poly}(n)$. The current state-of-art algorithm is due to Björklund, Husfeldt, Kaski and Koivisto [8], which is also an algebraic algorithm with running time $1.66^k\mathsf{poly}(n)$. All of

these algorithms are randomized. Our study of the k-BinaryTree problem, which is the problem of deciding whether a given graph G contains a binary tree of size at least k, is inspired by this line of results.

Several **NP**-hard problems are known to be solvable in specific families of graphs. Bipartite permutation graphs is one such family which is known to exhibit this behaviour [24, 33–35]. Our polynomial-time solvability result for these families of graphs crucially identifies the existence of structured optimal solutions to reduce the search space and solves the problem over this reduced search space.

1.2 Our contributions

1.2.1 Inapproximability results

Directed graphs. We first focus on directed graphs and in particular, on directed acyclic graphs. It is well-known that the longest path problem in DAGs is solvable in polynomial-time. In contrast, we show that DAGMaxBinaryTree does not even admit a constant-factor approximation. Furthermore, if DAGMaxBinaryTree admitted a polynomial-time $\exp\left(-O\left(\log n/\log\log n\right)\right)$-approximation algorithm then the Exponential Time Hypothesis would be violated.

▶ **Theorem 1.** *We have the following inapproximability results for DAGMaxBinaryTree on n-vertex input graphs:*

1. *DAGMaxBinaryTree does not admit a polynomial-time constant-factor approximation assuming* $\mathbf{P} \neq \mathbf{NP}$.
2. *If DAGMaxBinaryTree admits a polynomial-time* $\exp\left(-O\left(\log n/\log\log n\right)\right)$*-approximation, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(O\left(\sqrt{n}\right)\right)\right)$*, refuting the Exponential Time Hypothesis.*
3. *For any* $\varepsilon > 0$*, if DAGMaxBinaryTree admits a quasi-polynomial time* $\exp\left(-O\left(\log^{1-\varepsilon} n\right)\right)$*-approximation, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(\log^{O(1/\varepsilon)} n\right)\right)$*, thus refuting the Exponential Time Hypothesis.*

Remark. The longest path problem in DAGs can be solved using a linear program (LP) based on cut constraints. Based on this connection, an integer program (IP) based on cut constraints can be formulated for DAGMaxBinaryTree. In the full version of this work [11], we show that the LP-relaxation of this cut-constraints-based-IP has an integrality gap of $\Omega(n^{1/3})$ in n-vertex DAGs.

Undirected graphs. Next, we turn to undirected graphs. We show that UndirMaxBinaryTree does not have a constant-factor approximation and does not admit a quasi-polynomial-time $\exp(-O(\log^{0.63} n))$-approximation under the Exponential Time Hypothesis.

▶ **Theorem 2.** *We have the following inapproximability results for UndirMaxBinaryTree on n-vertex input graphs:*

1. *UndirMaxBinaryTree does not admit a polynomial-time constant-factor approximation assuming* $\mathbf{P} \neq \mathbf{NP}$.
2. *For* $c = \log_3 2$ *and any* $\varepsilon > 0$*, if UndirMaxBinaryTree admits a quasi-polynomial time* $\exp\left(-O\left(\log^{c-\varepsilon} n\right)\right)$*-approximation, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(\log^{O(1/\varepsilon)} n\right)\right)$*, thus refuting the Exponential Time Hypothesis.*

We summarize our hardness results for MBT on various graph families in Table 1 and contrast them with the corresponding known hardness results for the longest path problem on those families.

Table 1 Summary of inapproximability results. Here, n refers to the number of vertices in the input graph and ϵ is any positive constant. We include the known results for longest path for comparison. Text in gray refer to known results while text in black refer to our contributions.

Family	Assumption	Max Binary Tree	Longest Path
DAGs	$\mathbf{P \neq NP}$	No poly-time $\Omega(1)$-apx (Thm 1)	Poly-time solvable
	ETH	No poly-time $\exp(-O(\frac{\log n}{\log\log n}))$-apx No quasi-poly-time $\exp(-O(\log^{1-\varepsilon} n))$-apx (Thm 1)	Poly-time solvable
Directed	$\mathbf{P \neq NP}$	Same as DAGs	No poly-time $\frac{1}{n^{1-\varepsilon}}$-apx [9]
	ETH	Same as DAGs	Same as $\mathbf{P \neq NP}$
Undirected	$\mathbf{P \neq NP}$	No poly-time $\Omega(1)$-apx (Thm 2)	No poly-time $\Omega(1)$-apx [22]
	ETH	No quasi-poly-time $\exp(-O(\log^{0.63-\varepsilon} n))$-apx (Thm 2)	No quasi-poly-time $\exp(-O(\log^{1-\varepsilon} n))$-apx [22]

1.2.2 Algorithmic results

Fixed-parameter tractability. We denote the decision variant of MBT as k-BinaryTree– here the goal is to verify if a given graph contains a binary tree with at least k vertices. Since k-BinaryTree is **NP**-hard when k is part of the input, it is desirable to have an algorithm that runs in time $f(k)\mathsf{poly}(n)$ (i.e., a fixed parameter algorithm parameterized by the solution size). Our first algorithmic result achieves precisely this goal. Our algorithm is based on algebraic techniques.

▶ **Theorem 3.** *There exists a randomized algorithm that takes a directed graph $G = (V, E)$, a positive integer k, and a real value $\delta \in (0, 1)$ as input, runs in time $2^k\mathsf{poly}(|V|)\log(1/\delta)$ and*

1. *outputs "no" if G does not contain a binary tree of size k;*
2. *outputs a binary tree of size k with probability $1 - \delta$ if G contains one.*

Bipartite permutation graphs. Next, motivated by its connection to the max heapable subsequence problem, we study MBT in *bipartite permutation graphs*. A bipartite permutation graph is a permutation graph (undirected) which is also bipartite. We show that bipartite permutation graphs admit an efficient algorithm for MBT. Our algorithm exploits structural properties of bipartite permutation graphs. We believe that these structural properties could be helpful in solving MBT in permutation graphs which, in turn, could provide key insights towards solving MBT in permutation DAGs.

▶ **Theorem 4.** *There exists an algorithm to solve UndirMaxBinaryTree in n-vertex bipartite permutation graphs that runs in time $O(n^3)$.*

We summarize our algorithmic results for MBT in Table 2 and contrast them with the corresponding best known bounds for the longest path problem.

Table 2 Summary of algorithmic results. Here, n refers to the number of vertices in the input graph. We include the known results for longest path for comparison. Text in gray refer to known results while text in black refer to our contributions.

Problem	Max Binary Tree	Longest Path
FPT parameterized by solution size (Dir.)	$2^k\mathsf{poly}(n)$-time (Thm 3)	$1.66^k\mathsf{poly}(n)$-time [8]
Bipartite permutation graphs (Undir.)	$O(n^3)$-time (Thm 4)	$O(n)$-time [35]

We remark again that our inapproximability as well as algorithmic results are also applicable to the maximum degree-constrained tree problem for larger, but constant degree constraint. We focus on the degree constraint corresponding to binary trees for the sake of simplicity in exposition.

1.3 Proof techniques

In this section, we outline the techniques underlying our results.

1.3.1 Inapproximability results

At a very high level, our inapproximability results for MBT rely on the proof strategy for hardness of longest path due to Karger, Motwani, and Ramkumar [22], which has two main steps: (1) a *self-improving* reduction whose amplification implies that a constant-factor approximation immediately leads to a PTAS, and (2) a proof that there is no PTAS. However, we achieve both these steps in a completely different manner compared to the approach of Karger, Motwani, and Ramkumar. Both their steps are tailored for the longest path problem, but fail for the maximum degree-constrained tree problem. Our results for MBT require several novel ideas, as described next.

Karger, Motwani and Ramkumar's self-improving reduction for the longest path proceeds as follows: given an undirected graph G, they obtain a squared graph G^2 by replacing each edge $\{u, v\}$ of G with a copy of G by adding edges from u and v to all vertices in that edge copy. Let $OPT(G)$ be the length of the longest path in G. They make the following two observations: Obs (i) $OPT(G^2) \geq OPT(G)^2$ and Obs (ii) a path in G^2 of length at least $\alpha OPT(G^2)$ can be used to recover a path in G of length at least $\sqrt{\alpha}OPT(G)$. The first observation is because we can extend any path P in G into a path of length $|E(P)|^2$ by traversing each edge copy also along P. The second observation is because for any path P_2 in G^2 either P_2 restricted to some edge copy of G is a path of length at least $\sqrt{|E(P_2)|}$ or projecting P_2 to G (i.e., replacing each sub-path of P_2 in each edge copy by a single edge) gives a path of length at least $\sqrt{|E(P_2)|}$. We note that a similar construction of the squared graph for directed graphs also has the above mentioned observations: replace each directed arc (u, v) of G with a copy of G by adding arcs from u to all vertices in that edge copy and from all vertices in that edge copy to v.

In order to obtain inapproximability results for the maximum binary tree problem, we first introduce different constructions for the squared graph in the self-improving reduction compared to the ones by Karger et al. Moreover, our constructions of the squared graph differ substantially between undirected and directed graphs. Interestingly, our constructions also generalize naturally to the max degree-constrained tree problem. Secondly, although our reduction for showing the lack of PTAS in undirected graphs for MBT is also from TSP$(1, 2)$, it is completely different from that of Karger et al. and, once again, generalizes to the max degree-constrained tree problem. Thirdly, we show the lack of PTAS in DAGs for MBT by reducing from the max 3-coloring problem. This reduction is altogether new – the reader might recall that the longest path problem in DAGs is solvable in polynomial-time, so there cannot be a counterpart of this step (i.e., lack of PTAS in DAGs) for longest path. We next present further details underlying our proofs.

Self-improving reduction for directed graphs. We focus on the rooted variant of MBT in directed graphs. We first assume that the given graph G contains a source (if not, adding such a source vertex with arcs to all the vertices changes the optimum only by one). In

contrast to the squared graph described above (i.e., instead of adding edge copies), we replace every vertex in G by a copy of G (that we call as a vertex copy) and for every arc (u, v) in G, we add an arc from the root node of the vertex copy corresponding to u to the source node of the vertex copy corresponding to v. Finally, we declare the root node of the root vertex copy to be the root node of G^2. Let $\alpha \in (0, 1]$ and $OPT(G)$ be the number of vertices in the maximum binary tree in G. With this construction of the squared graph, we show that (1) $OPT(G^2) \geq OPT(G)^2$ and (2) an α-approximate rooted binary tree T_2 in G^2 can be used to recover a rooted binary tree T_1 in G which is a $\sqrt{\alpha}$-approximation. We emphasize that if G is a DAG, then the graph G^2 obtained by this construction is also a DAG.

Inapproximability for DAGs. In order to show the constant-factor inapproximability result for DAGs, it suffices to show that there is no PTAS (due to the self-improving reduction for directed graphs described above). We show the lack of a PTAS in DAGs by reducing from the max 3-coloring problem in 3-colorable graphs. It is known that this problem is APX-hard – in particular, there is no polynomial-time algorithm to find a coloring that colors at least 32/33-fraction of the edges properly [20]. Our reduction encodes the coloring problem into a DAGMaxBinaryTree instance in a way that recovers a consistent coloring for the vertices while also being proper for a large fraction of the edges. Our ETH-based inapproximability result is also a consequence of this reduction in conjunction with the self-improving reduction. We again emphasize that there is no counterpart of APX-hardness in DAGs for max binary tree in the longest path literature.

Self-improving reduction for undirected graphs. For UndirMaxBinaryTree, the self-improving reduction is more involved. Our above-mentioned reduction for DirMaxBinaryTree heavily exploits the directed nature of the graph (e.g., uses source vertices) and hence, is not applicable for undirected graphs. Moreover, the same choice of squared graph G^2 as Karger et al. [22] fails since Obs (ii) does not hold any more: the tree T_2 restricted to each edge copy may not be a tree (but it will be a forest). However, we observe that T_2 restricted to each edge copy may result in a forest with up to four binary trees in it. This observation and a more careful projection can be used to recover a tree of size at least $\sqrt{|V(T_2)|}/4$ (let us call this weakened Obs (ii)). Yet, weakened Obs (ii) is insufficient for a self-improving reduction. One approach to fix this would be to construct a *different squared graph* $G^{\boxtimes 2}$ that strengthens Obs (i) to guarantee that $OPT(G^{\boxtimes 2}) \geq 16OPT(G)^2$ while still allowing us to recover a binary tree of size $\sqrt{|V(T_2)|}/4$ in G from a binary tree T_2 in $G^{\boxtimes 2}$. Such a strengthened Obs (i) coupled with weakened Obs (ii) would complete the self-improving reduction. Our reduction is a variant of this approach: we introduce a construction of the squared-graph that strengthens Obs (i) by a factor of 2 while also weakening Obs (ii) only by a factor of 2. We prove these two properties of the construction by relying on a handshake-like property of binary trees which is a relationship between the number of nodes of each degree and the total number of nodes in the binary tree.

Inapproximability for undirected graphs. In order to show the constant-factor inapproximability result, it suffices to show that there is no PTAS (due to the self-improving reduction). We show the lack of a PTAS by reducing from TSP$(1, 2)$. We mention that Karger, Motwani, and Ramkumar [22] also show the lack of a PTAS for the longest path problem by reducing from TSP$(1, 2)$. However, our reduction is much different from their reduction. Our reduction mainly relies on the fact that if we add a pendant node to each vertex of a graph G and obtain a binary tree T that has a large number of such pendants, then the binary tree restricted to G cannot have too many nodes of degree three. Our ETH-based inapproximability result is also a consequence of this reduction in conjunction with the self-improving reduction.

1.3.2 Algorithmic results

A 2^kpoly(n) time algorithm for k-BinaryTree. The proof of this result is inspired by the algebrization technique introduced in [26, 27, 36] for designing randomized algorithms for k-Path and k-Tree – in k-Path, the goal is to recover a path of length k in the given graph while k-Tree asks to recover a *given tree* on k vertices in the given graph. Their idea is to encode a path (or the given tree) as a multilinear monomial term in a carefully constructed polynomial, which is efficiently computable using an arithmetic circuit. Then, a result due to Williams [36] is used to verify if the constructed polynomial contains a multilinear term – Williams' result gives an efficient randomized algorithm, which on input a small circuit that computes the polynomial, outputs "yes" if a multilinear term exists in the sum of products representation of the input polynomial, and "no" otherwise. The subgraph that is sought may then be extracted using an additional pass over the graph. Our main technical contribution is the construction of a polynomial P_G whose multilinear terms correspond to binary trees of size k in G and which is efficiently computable by an arithmetic circuit. We remark that the polynomial constructions in previous results do not readily generalize for our problem. Our key contribution is the construction of a suitable polynomial, based on a carefully designed recursion.

Efficient algorithm for bipartite permutation graphs. Our main structural insight for bipartite permutation graphs is that there exists a maximum binary tree which is *crossing-free* with respect to the so-called *strong ordering* of the vertices. With this insight, MBT in bipartite permutation graphs reduces to finding a maximum crossing-free binary tree. We solve this latter problem by dynamic programming.

Organization. We present the 2^kpoly(n) time algorithm for k-BinaryTree in Section 2. We present our hardness results for DAGs in Section 3. We conclude with a few open problems in Section 4. Due to page limits, we formulate an IP for DAGs and discuss its integrality gap, show our hardness results for undirected graphs, and design an efficient algorithm for bipartite permutation graphs in the full version [11].

1.4 Preliminaries

MBT in directed graphs. Given a directed graph $G = (V, E)$ and a vertex $r \in V$, we say that a subgraph T where $V(T) \subseteq V$ and $E(T) \subseteq E$, is an *r-rooted tree* in G if T is acyclic and every vertex v in T has a *unique* directed path (in T) to r. If the in-degree of each vertex in T is at most 2, then T is an *r-rooted binary tree*. The problem of interest in directed graphs is rooted-DirMaxBinaryTree: Given a directed graph $G = (V, E)$ and a root $r \in V$, the goal is to find an r-rooted binary tree in G with maximum number of vertices. The problem DAGMaxBinaryTree is a special case of rooted-DirMaxBinaryTree in which the input directed graph is a DAG. We recall that the rooted and unrooted variants of the maximum binary tree problem in DAGs are equivalent.

MBT in undirected graphs. Given an undirected graph $G = (V, E)$, we say that a subgraph T, where $V(T) \subseteq V$ and $E(T) \subseteq E$, is a *binary tree* in G if T is connected, acyclic, and $\deg_T(v) \leq 3$ for every vertex $v \in V(T)$. We will focus on the unrooted variant, i.e., UndirMaxBinaryTree, since the inapproximability results for the rooted variant are implied by inapproximability results for the unrooted variant. Here, we are given an undirected graph G and the goal is to find a binary tree in G with maximum number of vertices.

2 A $2^k\mathsf{poly}(n)$ time algorithm for k-BinaryTree

In this section, we present a randomized algorithm that solves k-BinaryTree exactly and runs in time $2^k\mathsf{poly}(n)$ where n is the number of vertices in the input graph. We recall that k-BinaryTree is the problem of deciding whether a given directed graph contains a binary tree of size k. Our algorithm is inspired by an algebraic approach for solving the k-Path problem – the algebraic approach relies on efficient detection of multilinear terms in a given polynomial.

k-Path, polynomials and multilinear terms. We begin with a recap of the algebraic approach to solve k-Path– here, the goal is to verify if a given (directed or undirected) graph G contains a path of length at least k. There has been a rich line of research dedicated to designing algorithms for k-Path with running time $\beta^k\mathsf{poly}(n)$ where $\beta > 1$ is a constant and n is the number of vertices in G (cf. [2, 8, 26, 36]). In particular, the algorithms in [26] and [36] are based on detecting multilinear terms in a polynomial.

We now recall the problem of detecting multilinear terms in a polynomial. Here, we are given a polynomial with coefficients in a finite field $\mathbb{F}_q$ and the goal is to verify if it has a multilinear term. We emphasize that the input polynomial is given *implicitly* by an arithmetic circuit consisting of additive and multiplicative gates. In other words, the algorithm is allowed to evaluate the polynomial at any point but does not have direct access to the sum-of-products expansion of the polynomial. We recall that a multilinear term in a polynomial $p \in \mathbb{F}_q[x_1, x_2, \cdots, x_m]$ is a monomial in the sum-of-products expansion of p consisting of only degree-1 variables. For example, in the following polynomial

$$p(x_1, x_2, x_3) = x_1^2x_2 + x_3 + x_1x_2x_3,$$

the monomials x_3 and $x_1x_2x_3$ are multilinear terms, whereas $x_1^2x_2$ is not a multilinear term since x_1 has degree 2. We will use the algorithm mentioned in the following theorem as a black box for detecting multilinear terms in a given polynomial.

▶ **Theorem 5** (Theorem 3.1 in [36]). *Let $P(x_1, \cdots, x_n)$ be a polynomial of degree at most k, represented by an arithmetic circuit of size $s(n)$ with additive gates (of unbounded fan-in), multiplicative gates (of fan-in two), and no scalar multiplications. There is a randomized algorithm that on input P runs in $2^k s(n)\cdot\mathsf{poly}(n)\log(1/\delta)$ time, outputs "yes" with probability $1-\delta$ if there is a multilinear term in the sum-product expansion of P, and outputs "no" if there is no multilinear term.*

The idea behind solving k-Path with the help of this theorem is to construct a polynomial p_G based on the input graph G so that p_G contains a multilinear term if and only if G contains a simple path of length k. At the same time, p_G should be computable by an arithmetic circuit of size $\mathsf{poly}(n)$. Koutis and Williams achieved these properties using the following polynomial:

$$p_G(x_1, \cdots, x_n) := \sum_{(v_{i_1}, v_{i_2}, \ldots, v_{i_k}): \text{ a walk in } G} x_{i_1}x_{i_2}\ldots x_{i_k}.$$

We recall that a walk in G is a sequence of vertices in which neighbouring vertices are adjacent in G. From the definition, it is easy to observe that there is a one-to-one correspondence between simple k-paths in G and multilinear terms in p_G. Moreover, it can be shown that there is an arithmetic circuit of size $O\left(k^2(m+n)\right)$ that computes p_G, where m is the number of edges and n is the number of vertices in G. See Chapter 10.4 of [14] for alternative constructions of this polynomial.

The polynomial construction for k-BinaryTree. Following the above-mentioned approach, we construct a polynomial P_G with the property that P_G contains a multilinear term if and only if G contains a binary tree of size k. Unfortunately, there is no immediate generalization of walks of length k that characterize binary trees on k vertices. So, instead of defining the polynomial conceptually, we will define the polynomial recursively by building the arithmetic circuit that computes P_G, and will prove the correspondence between multilinear terms in P_G and binary trees of size k in G. In the definition of our polynomial, we also need to introduce an auxiliary variable to eliminate low-degree multilinear terms in P_G (which is not an issue in the construction of the polynomial for k-Path).

Let $G = (V, E)$ be the given directed graph. For $v \in V$, let $\Delta_v^{in} \coloneqq \{u \in V : (u, v) \in E\}$. We begin by defining a polynomial $P_v^{(k)}$ for every $v \in V$ and every positive integer k, in $(n+1)$ variables $\{x_v\}_{v \in V} \cup \{y\}$:

$$P_v^{(k)} \coloneqq \begin{cases} x_v & \text{if } k = 1 \\ x_v \cdot y^{k-1} & \text{if } k > 1 \text{ and } \Delta_v^{in} = \varnothing \\ x_v \left(\sum_{u \in \Delta_v^{in}} P_u^{(k-1)} + \sum_{\ell=1}^{k-2} \left(\sum_{u_1 \in \Delta_v^{in}} P_{u_1}^{(\ell)} \right) \left(\sum_{u_2 \in \Delta_v^{in}} P_{u_2}^{(k-1-\ell)} \right) \right) & \text{if } k > 1 \text{ and } \Delta_v^{in} \neq \varnothing \end{cases}$$

Next, we define $P_G^{(k)} \coloneqq \sum_{v \in V} P_v^{(k)}$. We recall that a polynomial is homogenous if every monomial has the same degree. By induction on k, the polynomial $P_v^{(k)}$ is a degree-k homogeneous polynomial and so is $P_G^{(k)}$. Moreover, by the recursive definition, we see that $P_v^{(k)}$ can be represented as an arithmetic circuit of size $O(k^2 n)$ since there are kn polynomials in total, and computing each requires $O(1)$ addition gates (with unbounded fan-in) and $O(k)$ multiplication gates (with fan-in two). We show the following connection between multilinear terms in $P_G^{(k)}$ and binary trees in G.

▶ **Lemma 6.** *The graph G has a binary tree of size k rooted at r if and only if there is a multilinear term of the form $\prod_{v \in S} x_v$ in $P_r^{(k)}$ where $|S| = k$.*

Proof. We first show the forward direction, i.e., if G has a binary tree T of size k rooted at r, then there is a multilinear term of the form $\prod_{v \in T} x_v$ in $P_r^{(k)}$. We prove this by induction on k. The base case $k = 1$ follows since $P_r^{(1)} = x_r$. Suppose that the forward direction holds when $|T| \leq k - 1$. For $|T| = k$, we consider two cases.

1. The root r has only one child c. The subtree T_c of T rooted at c has size $k - 1$. By induction hypothesis there is a multilinear term $\prod_{v \in T_c} x_v$ in $P_c^{(k-1)}$. Since $c \in \Delta_r^{in}$, for some polynomial Q we can write

$$P_r^{(k)} = x_r \left(P_c^{(k-1)} + Q \right).$$

   Therefore $x_r \cdot \prod_{v \in T_c} x_v$ is a term in $P_r^{(k)}$. This term is multilinear and equals to $\prod_{v \in T} x_v$ since $r \notin T_c$.
2. The root r has two children c_1, c_2. Suppose that the subtree T_{c_1} rooted at c_1 has size ℓ, thus the subtree T_{c_2} rooted at c_2 has size $k - 1 - \ell$. The induction hypothesis implies that $P_{c_1}^{(\ell)}$ has a multilinear term $\prod_{v \in T_{c_1}} x_v$, and $P_{c_2}^{(k-1-\ell)}$ has a multilinear term $\prod_{v \in T_{c_2}} x_v$. Since $c_1, c_2 \in \Delta_r^{in}$, for some polynomial Q we can write

$$P_r^{(k)} = x_r \left(P_{c_1}^{(\ell)} P_{c_2}^{(k-1-\ell)} + Q \right).$$

   Therefore $x_r \left(\prod_{v \in T_{c_1}} x_v \right) \left(\prod_{v \in T_{c_2}} x_v \right)$ is a term in $P_r^{(k)}$. This term is multilinear and equals to $\prod_{v \in T} x_v$ because T is the disjoint union of r, T_{c_1} and T_{c_2}.

In both cases, the polynomial $P_r^{(k)}$ has a multilinear term $\prod_{v \in T} x_v$. This completes the inductive step.

Next, we show that if $P_r^{(k)}$ has a multilinear term of the form $\prod_{v \in S} x_v$ where $|S| = k$, then there is a binary tree T rooted at r in G with vertex set S. We prove this also by induction on k. The base case $k = 1$ is trivial since $P_r^{(1)} = x_r$ and there is a binary tree of size 1 rooted at r. Suppose that the statement holds for $k-1$ or less ($k > 1$).

Let $\prod_{v \in S} x_v$ be a multilinear term in $P_r^{(k)}$. We note that $r \in S$ since every term in $P_r^{(k)}$ contains x_r. Moreover, we may assume that $\Delta_r^{in} \neq \varnothing$ since otherwise $P_r^{(k)} = x_r \cdot y^{k-1}$ which does not contain any term of the form $\prod_{v \in S} x_v$. According to the definition of $P_r^{(k)}$, we could have two cases.

1. The term $\prod_{v \in S \setminus \{r\}} x_v$ is a multilinear term in $P_c^{(k-1)}$ for some $c \in \Delta_r^{in}$. The induction hypothesis implies that there is a binary tree T_c rooted at c with vertex set $S \setminus \{r\}$. Let T be the binary tree obtained by adding the edge (c, r) to T_c. Then T is a binary tree rooted at r with vertex set S.
2. The term $\prod_{v \in S \setminus \{r\}} x_v$ is a multilinear term in $P_{c_1}^{(\ell)} P_{c_2}^{(k-1-\ell)}$ for some $c_1, c_2 \in \Delta_r^{in}$ and some integer $1 \leq \ell \leq k-2$. In this case, since $P_{c_1}^{(\ell)}$ and $P_{c_2}^{(k-1-\ell)}$ are homogeneous polynomials of degree ℓ and $k-1-\ell$, we can partition $S \setminus \{r\}$ into two sets S_1 and S_2 with $|S_1| = \ell$ and $|S_2| = k-1-\ell$ such that $\prod_{v \in S_1} x_v$ is a multilinear term in $P_{c_1}^{(\ell)}$, and $\prod_{v \in S_2} x_v$ is a multilinear term in $P_{c_2}^{(\ell)}$. Applying the induction hypothesis, we obtain a binary tree T_{c_1} (rooted at c_1) with vertex set S_1 and a binary tree T_{c_2} (rooted at c_2) with vertex set S_2. Let T be the binary tree obtained by adding edges (c_1, r) and (c_2, r) to $T_{c_1} \cup T_{c_2}$. Then T is a binary tree rooted at r with vertex set $S_1 \cup S_2 \cup \{r\} = S$.

In both cases, we can find a binary tree T rooted at r with vertex set S. This completes the inductive step. ◀

With this choice of $P_G^{(k)}$, we call the algorithm appearing in Theorem 5 on input polynomial $\tilde{P}_G^{(k)} \coloneqq y \cdot P_G^{(k)}$, and output the result. We note that every multilinear term of the form $\prod_{v \in S} x_v$ in $P_G^{(k)}$ becomes a multilinear term of the form $y \cdot \prod_{v \in S} x_v$ in $\tilde{P}_G^{(k)}$, and every multilinear term of the form $y \cdot \prod_{v \in S} x_v$ in $P_G^{(k)}$ becomes $y^2 \cdot \prod_{v \in S} x_v$ in $\tilde{P}_G^{(k)}$, which is no longer a multilinear term. In light of Lemma 6, the graph G contains a binary tree of size k if and only if the degree-$(k+1)$ homogeneous polynomial $\tilde{P}_G^{(k)}$ has a multilinear term. The running time is $2^{k+1} \cdot O(k^2 n) \cdot \mathsf{poly}(n+1) \log(1/\delta) = 2^k \cdot \mathsf{poly}(n) \log(1/\delta)$.

We remark that this algorithm does not immediately tell us the tree T (namely the edges in T). However, we can find the edges in T with high probability via a reduction from the search variant to the decision variant. This is formalized in the next lemma.

▶ **Lemma 7.** *Suppose that there is an algorithm $\mathcal{A}$ which takes as input a directed graph $G = (V, E)$, an integer k and $\delta' \in (0,1)$ runs in time $2^k \mathsf{poly}(|V|) \log(1/\delta')$ and*

- *outputs "yes" with probability at least $1 - \delta'$ if G contains a binary tree of size k,*
- *outputs "no" with probability 1 if G does not contain a binary tree of size k.*

Then there also exists an algorithm $\mathcal{A}'$ which for every $\delta \in (0,1)$ outputs a binary tree T of size k with probability at least $1 - \delta$ when the answer is "yes", and runs in time $2^k \mathsf{poly}(|V|) \log(1/\delta)$.

Proof. The algorithm $\mathcal{A}'$ iterates through all arcs $e \in E$ and calls $\mathcal{A}$ on $(G - e, k)$ with $\delta' = \delta/m$ where $G - e = (V, E \setminus \{e\})$ and $m = |E|$. If for some $e \in E$ the call to $\mathcal{A}$ outputs "yes", we remove e from G (i.e., set $G \leftarrow G - e$) and continue the process. We will show that when the algorithm terminates, the arcs in G constitute a binary tree of size k (if there exists one) with probability at least $1 - \delta$.

Suppose the order in which $\mathcal{A}'$ processes the arcs is $e_1, e_2, \cdots, e_m$, and the graph at iteration t is denoted by $G^{(t)}$. Let B_t denote the event "$G^{(t-1)} - e_t$ contains a binary tree of size k, but the call to $\mathcal{A}\left(G^{(t-1)} - e_t, k\right)$ returns no". Due to the assumption we made for $\mathcal{A}$, event B_t happens with probability at most δ'. Since the algorithm $\mathcal{A}$ has perfect soundness, whenever $\mathcal{A}'$ removes an edge we are certain that the remaining graph still contains a binary tree of size k (otherwise the call to $\mathcal{A}$ would never return "yes"). That means if $G^{(0)} = G$ contains a binary tree of size k then $G^{(t)}$ contains a binary tree of size k for all $0 \leq t \leq m$. Therefore if none of the events B_t happens, the final graph $G^{(m)}$ is a binary tree of size k. The probability of failure is upper bounded by

$$\Pr\left[\bigcup_{t=1}^{m} B_t\right] \leq m \cdot \delta' = m \cdot \frac{\delta}{m} = \delta.$$

Since algorithm $\mathcal{A}'$ makes m calls to algorithm $\mathcal{A}$, the running time of $\mathcal{A}'$ is $m \cdot 2^k \mathsf{poly}\left(|V|\right) \log\left(1/\delta'\right) = 2^k \mathsf{poly}\left(|V|\right) \log\left(1/\delta\right)$. ◀

Theorem 5 in conjunction with Lemmas 6 and 7 complete the proof of Theorem 3.

3 Hardness results for DAGs

In this section, we show the inapproximability of finding a maximum binary tree in DAGs. The *size* of a binary tree denotes the number of vertices in the tree.

3.1 Self-improvability for directed graphs

We show that an algorithm for ROOTED-DIRMAXBINARYTREE achieving a constant factor approximation can be used to design a PTAS in Theorem 11. We emphasize that this result holds for arbitrary directed graphs and not just DAGs. The idea is to gradually boost up the approximation ratio by running the constant-factor approximation algorithm on squared graphs. Our notion of *squared graph* will be the following.

▶ **Definition 8.** *Given a directed graph $G = (V, E)$ with root r, the* squared graph G^2 *is the directed graph obtained by performing the following operations on G:*

1. *Construct $G' = (V', E')$ by introducing a source vertex s, i.e., $V' := V \cup \{s\}$. We add arcs from s to every vertex in G, i.e., $E' := E \cup \{(s, v) : v \in V\}$.*
2. *For each $u \in V$ (we note that V does not include the source vertex), we create a copy of G' that we denote as a* vertex copy G'_u. *We will denote the root vertex of G'_u by r_u, and the source vertex of G'_u by s_u.*
3. *For each $(u, v) \in E$, we create an arc (r_u, s_v).*
4. *We declare the root of G^2 to be r_r, i.e. the root vertex of the vertex copy G'_r.*

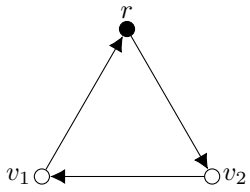
We define $G^{2^{k+1}}$ recursively as $G^{2^{k+1}} := \left(G^{2^k}\right)^2$ with the base case $G^1 := G$.

Given a directed graph G with $n-1$ vertices, the number of vertices in G^{2^k} satisfies the recurrence relation

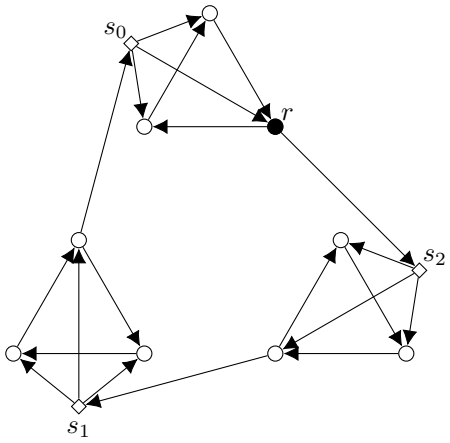
$$\left|V\left(G^{2^k}\right)\right| = \left|V\left(G^{2^{k-1}}\right)\right| \cdot \left(\left|V\left(G^{2^{k-1}}\right)\right| + 1\right) = \left|V\left(G^{2^{k-1}}\right)\right|^2 + \left|V\left(G^{2^{k-1}}\right)\right|.$$

Hence, we have

$$\left|V\left(G^{2^k}\right)\right| + 1 \leq \left(\left|V\left(G^{2^{k-1}}\right)\right| + 1\right)^2 \leq \left(\left|V\left(G^{2^{k-2}}\right)\right| + 1\right)^{2^2} \leq \cdots \leq \left(\left|V\left(G^{2^0}\right)\right| + 1\right)^{2^k} = n^{2^k}.$$



(a) G rooted at the black node.



(b) G^2 rooted at the black node. Source nodes are represented by diamonds.

Figure 1 Directed Squared Graph.

We use $OPT(G)$ to denote the size (number of vertices) of a maximum binary tree in G. The following lemma shows that $OPT(G)$ is super-multiplicative under the squaring operation.

▶ **Lemma 9.** *For any fixed root r, $OPT(G^2) \geq OPT(G)^2$.*

Proof. Suppose we have an optimal r-rooted binary tree T_1 of G, i.e. $|V(T_1)| = OPT(G)$. We construct an r_r-rooted binary tree T_2 of G^2 as follows:

1. For $v \in V(G)$, define $T'_v = T_v \cup \{s_v\}$ to be the optimal r_v-rooted binary tree in the vertex copy G'_v where T_v is identical to T_1 and the source vertex s_v is connected to an arbitrary leaf node in T_v.
2. For every vertex $v \in T_1$, add T'_v to T_2. This step generates $|V(T_1)| \cdot (|V(T_1)| + 1)$ vertices in T_2.
3. Connect the copies selected in step 2 by adding the arc (r_u, s_v) to T_2 for every arc $(u, v) \in T_1$.

Since T_1 is an r-rooted binary tree (in G), it follows that T_2 is an r_r-rooted binary tree (in G^2). Moreover, the size of T_2 is

$$|V(T_2)| = |V(T_1)| \cdot (|V(T_1)| + 1) \geq OPT(G)^2,$$

which cannot exceed $OPT\left(G^2\right)$. ◀

The following lemma shows that a large binary tree in G^2 can be used to obtain a large binary tree in G.

▶ **Lemma 10.** *For every $\alpha \in (0, 1]$, given an r_r-rooted binary tree T_2 in G^2 with size*

$$|V(T_2)| \geq \alpha OPT\left(G^2\right) - 1,$$

there is a linear-time (in the size of G^2) algorithm that finds an r-rooted binary tree T_1 of G with size

$$|V(T_1)| \geq \sqrt{\alpha} OPT(G) - 1.$$

Proof. Let $U \coloneqq \{v\colon v \in V(G) \text{ such that } r_v \in V(T_2)\}$ and $A \coloneqq \{(v,w) : v,w \in V(G), (r_v, s_w) \in E(T_2)\}$. We note that $T_1' := (U, A)$ is an r-rooted binary tree in G. This is because the path from every $v \in U$ to the root r is preserved, and the in-degree of every node $w \in U$ is bounded by the in-degree of s_w (in T_2), which is thus at most 2, and similarly the out-degree of every node is at most 1. We also remark that T_1' can be found in linear time. If $|U| \geq \sqrt{\alpha}OPT(G) > \sqrt{\alpha}OPT(G) - 1$, then the lemma is already proved. So, we may assume that $|U| < \sqrt{\alpha}OPT(G)$.

We now consider $T_v' \coloneqq (V(T_2) \cap V(G_v'), E(T_2) \cap E(G_v'))$ for $v \in U$. We can view T_v' as the restriction of T_2 to G_v', hence every node of T_v' has out-degree at most 2. Since T_2 is an r_r-rooted binary tree in G^2, every vertex in $V(T_2) \cap V(G_v')$ has a unique directed path (in T_2) to r_r, which must go through r_v, thus every vertex in $V(T_2) \cap V(G_v')$ has a unique directed path to r_v. It follows that T_v' is an r_v-rooted binary tree in the vertex copy G_v'.

We now show that there exists $v \in U$ such that $|V(T_v')| \geq \sqrt{\alpha}OPT(G)$. Suppose not, which means for every $v \in U$ we have $|V(T_v')| < \sqrt{\alpha}OPT(G)$. Then

$$\begin{aligned} |V(T_2)| &= \sum_{v \in U} |V(T_v')| < \sum_{v \in U} \left(\sqrt{\alpha}OPT(G)\right) < \sqrt{\alpha}OPT(G) \cdot \sqrt{\alpha}OPT(G) \\ &= \alpha OPT(G)^2 \leq \alpha \cdot OPT\left(G^2\right), \end{aligned}$$

a contradiction. The last inequality is due to Lemma 9.

In linear time we can find a binary tree T_v' with the desired size $|V(T_v')| \geq \sqrt{\alpha}OPT(G)$. To complete the proof of the lemma, we let $T_1 \coloneqq T_v' \setminus \{s_v\}$ which is (isomorphic to) an r-rooted binary tree in G with size at least $\sqrt{\alpha}OPT(G) - 1$. ◀

▶ **Theorem 11.** *If* rooted-DirMaxBinaryTree *has a polynomial-time algorithm that achieves a constant-factor approximation, then it has a PTAS.*

Proof. Suppose that we have a polynomial-time algorithm $\mathcal{A}$ that achieves an α-approximation for rooted-DirMaxBinaryTree. Given a directed graph G, root r and $\varepsilon > 0$, let

$$k \coloneqq 1 + \left\lceil \log_2 \frac{\log_2 \alpha}{\log_2(1-\varepsilon)} \right\rceil$$

be an integer constant that depends on α and ε. We construct G^{2^k} and run algorithm $\mathcal{A}$ on G^{2^k}. Then, we get a binary tree in G^{2^k} of size at least $\alpha OPT\left(G^{2^k}\right) - 1$. By Lemma 10, we can obtain an r-rooted binary tree in G of size at least

$$\alpha^{2^{-k}} OPT(G) - 1 \geq \alpha^{2^{-k+1}} OPT(G) \geq (1-\varepsilon) OPT(G).$$

The first inequality holds as long as

$$OPT(G) \geq \frac{1}{\sqrt{1-\varepsilon} - (1-\varepsilon)} \geq \frac{1}{\alpha^{2^{-k}} - \alpha^{2^{-k+1}}}.$$

We note that if $OPT(G)$ is smaller than $1/\left(\alpha^{2^{-k}} - \alpha^{2^{-k+1}}\right)$ which is a constant, then we can solve the problem exactly by brute force in polynomial time. Finally, we also observe that for fixed ε, the running time of this algorithm is polynomial since there are at most $n^{2^k} = n^{O(1)}$ vertices in the graph G^{2^k}. ◀

3.2 APX-hardness for DAGs

Next, we show the inapproximability results for DAGs. We begin by recalling DAGMaxBinaryTree: We begin by recalling the problem:

DAGMaxBinaryTree
Given: A directed acyclic graph $G = (V, E)$ and a root $r \in V$.
Goal: An r-rooted binary tree in G with the largest number of nodes.

We may assume that the root is the only vertex that has no outgoing arcs as we may discard all vertices that cannot reach the root. We show that DAGMaxBinaryTree is APX-hard by reducing from the following problem.

Max-3-Colorable-Subgraph
Given: An undirected graph G that is 3-colorable.
Goal: A 3-coloring of G that maximizes the fraction of properly colored edges.

It is known that finding a 3-coloring that properly colors at least 32/33-fraction of edges in a given 3-colorable graph is NP-hard [5,20]. In particular, Max-3-Colorable-Subgraph is APX-hard. We reduce Max-3-Colorable-Subgraph to DAGMaxBinaryTree. Let $G = (V, E)$ be the input 3-colorable undirected graph with $n := |V|$ and $m := |E|$. For $\varepsilon > 0$ to be fixed later, we construct a DAG, denoted $D(G, \varepsilon)$, as follows (see Figure 2 for an illustration):

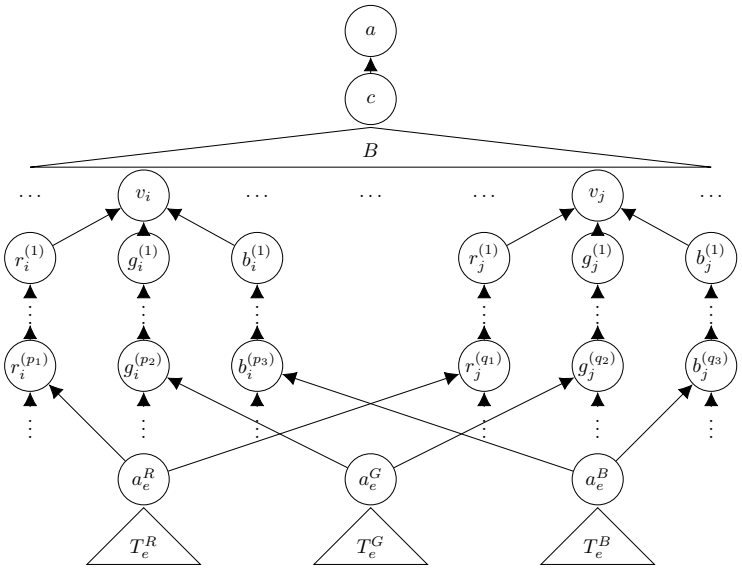


Figure 2 DAG $D(G, \varepsilon)$ constructed in the reduction from Max-3-Colorable-Subgraph to DAGMaxBinaryTree.

1. Create a directed binary tree B rooted at node c with $n := |V|$ leaf nodes. We will identify each leaf node by a unique vertex $v \in V$. Create a super root a and arc $c \to a$. This tree and the super root would have $2n$ nodes, including the super root node a, n leaf nodes, and $n - 1$ internal nodes.
2. For every $i \in V$, we introduce three directed paths of length n that will be referred to as R_i, G_i and B_i. Let R_i be structured as $r_i^{(1)} \leftarrow r_i^{(2)} \leftarrow \cdots \leftarrow r_i^{(n)}$, and similarly introduce $g_i^{(k)}$ and $b_i^{(k)}$ with the same structure. Also add arcs $r_i^{(1)} \to v_i$, $g_i^{(1)} \to v_i$ and $b_i^{(1)} \to v_i$.

3. For every edge $e = \{i, j\} \in E$, introduce three directed binary trees that will be referred to as T_e^R, T_e^G, and T_e^B, each with $t = \left\lceil \frac{2\varepsilon n(n+1)+4n^2}{\varepsilon m} \right\rceil$ nodes. Let the roots of the binary trees T_e^R, T_e^G, and T_e^B be a_e^R, a_e^G, and a_e^B respectively. Add arcs $a_e^R \to r_i^{(p_1)}$ and $a_e^R \to r_j^{(q_1)}$ where $r_i^{(p_1)}$ and $r_j^{(q_1)}$ are two nodes in R_i and R_j with in-degree strictly smaller than 2. We note that R_i is a path with n nodes so such a node always exists. Similarly connect a_e^G to $g_i^{(p_2)}$ and $g_j^{(q_2)}$, and a_e^B to $b_i^{(p_3)}$ and $b_j^{(q_3)}$ in the directed paths G_i and B_i, respectively.

The constructed graph $D(G, \varepsilon)$ is a DAG. We fix a to be the root. The number of nodes N in $D(G, \varepsilon)$ is $N = 3mt + 3n \cdot n + 2n = 3mt + 3n^2 + 2n$. We note that every node $v_i \in V$ has in-degree exactly 2 in every a-rooted maximal binary tree in $D(G, \varepsilon)$. The idea of this reduction is to encode the color of v_i as the unique path among R_i, G_i, B_i that is *not* in the subtree under v_i. The following two lemmas summarize the main properties of the DAG constructed above.

▶ **Lemma 12.** *Let T be a maximal a-rooted binary tree of $D(G, \varepsilon)$. If $|V(T)| \geq (1-\varepsilon/4)(N - n^2)$, then at most εm nodes among $\cup_{e \in E}\{a_e^R, a_e^G, a_e^B\}$ are not in T.*

Proof. Suppose more than εm such nodes are missing from T. For each node a_e^R that is not in T, the corresponding subtree T_e^R is also not in T (same for a_e^G and a_e^B). Therefore

$$|V(T)| < N - \varepsilon mt = 3mt + 3n^2 + 2n - \varepsilon mt = \left(1 - \frac{\varepsilon}{4}\right) \cdot 3mt + 3n^2 + 2n - \frac{\varepsilon}{4} mt.$$

The choice of t implies that $\varepsilon mt/4 > \varepsilon n(n+1)/2 + n^2$. Therefore

$$\begin{aligned} |V(T)| &< \left(1 - \frac{\varepsilon}{4}\right) \cdot 3mt + 3n^2 + 2n - \frac{\varepsilon n(n+1)}{2} - n^2 \\ &< \left(1 - \frac{\varepsilon}{4}\right) \cdot 3mt + \left(1 - \frac{\varepsilon}{4}\right)(2n^2 + 2n) \\ &= \left(1 - \frac{\varepsilon}{4}\right)(N - n^2), \end{aligned}$$

a contradiction. ◀

▶ **Lemma 13.** *If G is 3-colorable, then every a-rooted maximum binary tree in $D(G, \varepsilon)$ has size exactly $N - n^2$.*

Proof. We first note that every binary subtree of $D(G, \varepsilon)$ has size at most $N - n^2$. This is because there are n vertices with in-degree 3 (namely $v_1, v_2, \cdots, v_n$). For each such vertex v_i, there are 3 vertices $r_i^{(1)}, g_i^{(1)}$ and $b_i^{(1)}$ whose only outgoing arc is to v_i. Moreover, each vertex $r_i^{(1)}$ (and similarly $g_i^{(1)}$ and $b_i^{(1)}$) is the end-vertex of an induced path of length n.

Suppose G is 3-colorable. We now construct an a-rooted binary tree T of size $N - n^2$ in $D(G, \varepsilon)$. We focus on the nodes to be discarded so that we may construct a binary spanning tree with the remaining nodes. Let $\sigma \colon V \to \{Red, Green, Blue\}$ be a proper 3-coloring of G. If $\sigma(v_i) = Red$, we discard the path R_i. The cases where $\sigma(v_i) \in \{Green, Blue\}$ are similar. Since there are no monochromatic edges, there do not exist $e = \{v_i, v_j\} \in E$ and $C \in \{R, G, B\}$ such that both parents of a_e^C are not in T. Therefore every binary tree T_e^C is contained as a subtree in T. ◀

▶ **Theorem 14.** *Suppose there is a PTAS for DAGMaxBinaryTree on DAGs, then for every $\varepsilon > 0$ there is a polynomial-time algorithm which takes as input an undirected 3-colorable graph G, and outputs a 3-coloring of G that properly colors at least $(1 - \varepsilon)m$ edges.*

Proof. Let $G = (V, E)$ be the given undirected 3-colorable graph. We construct $D(G, \varepsilon)$ in polynomial time. We note that the constructed graph $D(G, \varepsilon)$ is a directed acyclic graph. We now run the PTAS for DAGMaxBinaryTree on $D(G, \varepsilon)$ and root a to obtain a $(1 - \varepsilon/4)$-approximate maximum binary tree in $D(G, \varepsilon)$. By Lemma 13 and the fact that G is 3-colorable, the PTAS will output an a-rooted binary tree T of size at least

$$\left(1 - \frac{\varepsilon}{4}\right)(N - n^2).$$

We may assume that T is a maximal binary tree in $D(G, \varepsilon)$ (if not, then add more vertices to T until we cannot add any further). Maximality ensures that the nodes v_i are in the tree T and moreover, the in-degree of v_i in T is exactly 2. For each $v_i \in V$, let c_i be the unique node among $\left\{r_i^{(1)}, g_i^{(1)}, b_i^{(1)}\right\}$ that is not in T. We define a coloring $\sigma : V \to \{Red, Green, Blue\}$ of G as

$$\forall v_i \in V, \quad \sigma(v_i) = \begin{cases} Red & \text{if } c_i = r_i^{(1)} \\ Green & \text{if } c_i = g_i^{(1)} \\ Blue & \text{if } c_i = b_i^{(1)}. \end{cases}$$

We now argue that the coloring is proper for at least $(1 - \varepsilon)$-fraction of the edges of G. Suppose we have an edge $e = \{v_i, v_j\}$ which is monochromatic under σ, and suppose w.l.o.g. $\sigma(v_i) = \sigma(v_j) = Red$. This means that neither $r_i^{(1)}$ nor $r_j^{(1)}$ is included in T. Therefore $a_e^R \notin T$ since neither of the two vertices with incoming arcs from a_e^R are in T. By Lemma 12, we know that at most εm vertices among $\cup_{e \in E}\{a_e^R, a_e^G, a_e^B\}$ can be excluded from T. Hence, the coloring σ that we obtained can violate at most εm edges in G. ◀

Finally, we prove Theorem 1 using the self-improving argument (Theorem 11) and the APX-hardness of DAGMaxBinaryTree (Theorem 14).

Proof of Theorem 1.

1. We observe that the graph G^2 constructed in Section 3 for the self-improving reduction is a DAG if G is a DAG. Therefore, by Theorem 11, a polynomial-time constant-factor approximation for DAGMaxBinaryTree would imply a PTAS for DAGMaxBinaryTree, a contradiction to APX-hardness shown in Theorem 14.
2. Next we show hardness under the Exponential Time Hypothesis. Suppose there is a polynomial-time algorithm $\mathcal{A}$ for DAGMaxBinaryTree that achieves an $\exp\left(-C \cdot \log_2 n / \log_2 \log_2 n\right)$-approximation for some constant $C > 0$. Given the input graph G with $n - 1$ vertices, let k be an integer that satisfies

   $$2^{\sqrt{n}} \leq n^{2^k} \leq 2^{2\sqrt{n}},$$

   and run $\mathcal{A}$ on G^{2^k} to obtain a binary tree with size at least

   $$\exp\left(-C \cdot \log_2 N / \log_2 \log_2 N\right) OPT\left(G^{2^k}\right),$$

   where $N = n^{2^k}$ upper bounds the size of G^{2^k}. Recursively running the algorithm suggested in Theorem 11 k times gives us a binary tree in G with size at least

   $$\begin{aligned} &\exp\left(-C \cdot \frac{\log_2 N}{\log_2 \log_2 N \cdot 2^k}\right) OPT(G) - 1 \\ \geq &\exp\left(-C \cdot \frac{2\sqrt{n}}{\log_2 \sqrt{n}} \cdot \frac{\log_2 n}{\sqrt{n}}\right) OPT(G) - 1 \\ \geq &\exp\left(-4C\right) OPT(G) - 1 \geq \frac{1}{2} \cdot \exp\left(-4C\right) OPT(G). \end{aligned}$$

The last inequality holds as long as

$$OPT(G) \geq 2 \cdot e^{4C}.$$

We note that if $OPT(G)$ is smaller than $2e^{4C}$ which is a constant, we can solve the problem exactly by brute force in polynomial time. Otherwise the above procedure can be regarded as a constant-factor approximation for DAGMaxBinaryTree. The running time is polynomial in

$$N = n^{2^k} = \exp\left(O\left(\sqrt{n}\right)\right),$$

which is sub-exponential. Moreover, from item 1 we know that it is **NP**-hard to approximate DAGMaxBinaryTree within a constant factor, thus $\mathbf{NP} \subseteq \mathbf{DTIME}\left(\exp\left(O\left(\sqrt{n}\right)\right)\right)$.

3. The proof of this item is almost identical to the previous one except that we choose a different integer k. Suppose there is an algorithm $\mathcal{A}'$ for DAGMaxBinaryTree that achieves a $\exp\left(-C \cdot \log^{1-\varepsilon} n\right)$-approximation for some constant $C > 0$, and runs in time $\exp\left(O\left(\log^d n\right)\right)$ for some constant $d > 0$. We show that there is an algorithm that achieves a constant-factor approximation for DAGMaxBinaryTree, and runs in time $\exp\left(O\left(\log^{d/\varepsilon} n\right)\right)$.
Given a DAG G on $n-1$ vertices as input for DAGMaxBinaryTree, let $k = \left\lceil \left(\frac{1}{\varepsilon} - 1\right) \log_2 \log n \right\rceil$ be an integer that satisfies

$$\left(2^k \log n\right)^{1-\varepsilon} \leq 2^k \leq 2 \left(\log n\right)^{\frac{1}{\varepsilon}-1}.$$

Running $\mathcal{A}'$ on G^{2^k} gives us a binary tree with size at least

$$\exp\left(-C \cdot \log^{1-\varepsilon} N\right) OPT\left(G^{2^k}\right),$$

where $N = n^{2^k}$ upper bounds the size of G^{2^k}. Recursively running the algorithm suggested in Theorem 11 k times gives us a binary tree in G with size at least

$$\begin{aligned}
&\exp\left(-C \cdot \frac{\log^{1-\varepsilon} N}{2^k}\right) OPT(G) - 1 \\
\geq &\exp\left(-C \cdot \frac{\left(2^k \log n\right)^{1-\varepsilon}}{2^k}\right) OPT(G) - 1 \\
\geq &\exp\left(-C\right) OPT(G) - 1 \geq \frac{1}{2} \cdot \exp\left(-C\right) OPT(G).
\end{aligned}$$

The last inequality holds as long as

$$OPT(G) \geq 2 \cdot e^{C}.$$

We note that if $OPT(G)$ is smaller than $2e^{C}$ which is a constant, we can solve the problem exactly by brute force in polynomial time. Otherwise the above procedure can be regarded as a constant-factor approximation for DAGMaxBinaryTree. The running time is quasi-polynomial in N, i.e. for some constant $C' > 0$, the running time is upper-bounded by

$$\exp\left(C'\left(\log^d N\right)\right) = \exp\left(C'\left(\left(2^k \log n\right)^d\right)\right) \leq \exp\left(C'\left(\log^{d/\varepsilon} n\right)\right). \quad \blacktriangleleft$$

4 Conclusion and Open Problems

In this work, we introduced the maximum binary tree problem (MBT) and presented hardness of approximation results for undirected, directed, and directed acyclic graphs, a fixed-parameter algorithm with the solution as the parameter, and efficient algorithms for bipartite permutation graphs. Our work raises several open questions that we state below.

Inapproximability of DirMaxBinaryTree. The view that MBT is a variant of the longest path problem leads to the natural question of whether the inapproximability results for MBT match that of longest path: Is MBT in directed graphs (or even in DAGs) hard to approximate within a factor of $1/n^{1-\varepsilon}$ (we recall that longest path is hard to approximate within a factor of $1/n^{1-\varepsilon}$ [9])? We remark that the self-improving technique is weak to handle $1/n^{1-\varepsilon}$-approximations since the squaring operation yields no improvement. The reduction in [9] showing $1/n^{1-\varepsilon}$-inapproximability of longest paths is from a restricted version of the vertex-disjoint paths problem and is very specific to paths. Furthermore, directed cycles play a crucial role in their reduction for a fundamental reason: longest path is polynomial-time solvable in DAGs. However, it is unclear if directed cycles are the source of hardness for MBT in digraphs (since MBT is already hard in DAGs).

Bicriteria Approximations. Given our inapproximability results, one natural algorithmic possibility is that of bicriteria approximations: can we find a tree with at least $\alpha \cdot OPT$ vertices while violating the degree bound by a factor of at most β? In particular, this motivates an intriguing direction concerning the longest path problem: Given an undirected/directed graph G with a path of length k, can we find a c_1-degree tree in G with at least k/c_2 vertices for some constants c_1 and c_2 efficiently?

Maximum Binary Tree in Permutation DAGs. Finally, it would be interesting to resolve the complexity of MBT in permutation DAGs (and permutation graphs). This would also resolve the open problem posed by Byers, Heeringa, Mitzenmacher, and Zervas of whether the maximum heapable subsequence problem is solvable in polynomial time [10].

References

1. Louigi Addario-Berry, Ketan Dalal, and Bruce A Reed. Degree constrained subgraphs. *Electronic Notes in Discrete Mathematics*, 19:257–263, 2005.
2. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
3. Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. Degree-constrained subgraph problems: Hardness and approximation results. In *Approximation and Online Algorithms*, pages 29–42, 2009.
4. Omid Amini, Ignasi Sau, and Saket Saurabh. Parameterized complexity of the smallest degree-constrained subgraph problem. In *Parameterized and Exact Computation*, pages 13–29, 2008.
5. Per Austrin, Ryan O'Donnell, and John Wright. A new point of NP-hardness for 2-to-1 Label-Cover. In *Proceedings of the 15th Annual International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, APPROX '12, pages 1–12, 2012.
6. János Balogh, Cosmin Bonchiş, Diana Diniş, Gabriel Istrate, and Ioan Todinca. On the heapability of finite partial orders. *Discrete Mathematics and Theoretical Computer Science*, 22(1):paper # 17, 2020.
7. Nikhil Bansal, Rohit Khandekar, and Viswanath Nagarajan. Additive guarantees for degree-bounded directed network design. *SIAM J. Comput.*, 39(4):1413–1431, October 2009.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017.

9 Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. Approximating longest directed paths and cycles. In *Automata, Languages and Programming*, pages 222–233, 2004.

10 John Byers, Brent Heeringa, Michael Mitzenmacher, and Georgios Zervas. Heapable sequences and subseqeuences. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics*, ANALCO '11, pages 33–44, 2011.

11 Karthekeyan Chandrasekaran, Elena Grigorescu, Gabriel Istrate, Shubhang Kulkarni, Young-San Lin, and Minshen Zhu. The maximum binary tree problem. *arXiv preprint*, 2019. arXiv:1909.07915.

12 Kamalika Chaudhuri, Satish Rao, Samantha Riesenfeld, and Kunal Talwar. A push–relabel approximation algorithm for approximating the minimum-degree mst problem and its generalization to matroids. *Theoretical Computer Science*, 410(44):4489–4503, 2009.

13 Kamalika Chaudhuri, Satish Rao, Samantha Riesenfeld, and Kunal Talwar. What Would Edmonds Do? Augmenting Paths and Witnesses for Degree-Bounded MSTs. *Algorithmica*, 55(1):157–189, September 2009.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

15 Paul Erdös, Ralph J Faudree, CC Rousseau, and RH Schelp. Subgraphs of minimal degree k. *Discrete Math*, 85(1):53–58, 1990.

16 Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994. doi:10.1006/jagm.1994.1042.

17 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 448–456, 1983.

18 Michael Garey and David Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

19 Michel X. Goemans. Minimum bounded degree spanning trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 273–282, 2006.

20 Venkatesan Guruswami and Ali Kemal Sinop. Improved inapproximability results for maximum k-colorable subgraph. *Theory of Computing*, 9:413–435, 2013.

21 Gabriel Istrate and Cosmin Bonchiş. Heapability, interactive particle systems, partial orders: Results and open problems. In *Proceedings of DCFS'2016, 18th International Conference on Descriptional Complexity of Formal Systems*, pages 18–28. Springer, 2016.

22 David R. Karger, Rajeev Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

23 Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. On some network design problems with degree constraints. *Journal of Computer and System Sciences*, 79(5):725–736, 2013.

24 Ton Kloks, Dieter Kratsch, and Haiko Müller. Bandwidth of chain graphs. *Information Processing Letters*, 68(6):313–315, 1998.

25 Jochen Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM J. Comput.*, 31(6):1783–1793, June 2002.

26 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *International Colloquium on Automata, Languages, and Programming*, ICALP '08, pages 575–586, 2008.

27 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *International Colloquium on Automata, Languages, and Programming*, ICALP '09, pages 653–664, 2009.

28 Jochen Könemann and R. Ravi. Primal-dual meets local search: Approximating msts with nonuniform degree bounds. *SIAM Journal on Computing*, 34(3):763–773, 2005.

29 Lap Chi Lau, Joseph (Seffi) Naor, Mohammad Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM Journal on Computing*, 39(3):1062–1087, 2009.

30 Jaclyn Porfilio. A combinatorial characterization of heapability. Master's thesis, Williams College, 2015.

31 R. Ravi, Madhav Marathe, S. S. Ravi, Daniel Rosenkrantz, and Harry B. Hunt III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31(1):58–78, September 2001.

32 Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *J. ACM*, 62(1):1–19, March 2015.

33 Jacqueline Smith. Minimum degree spanning trees on bipartite permutation graphs. Master's thesis, University of Alberta, 2011.

34 Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

35 Ryuhei Uehara and Yushi Uno. Efficient algorithms for the longest path problem. In *Proceedings of the 15th International Conference on Algorithms and Computation*, ISAAC '04, pages 871–883, 2004.

36 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.