Fast Algorithms for Pseudoarboricity

Markus Blumenstock*

Abstract

The densest subgraph problem, which asks for a subgraph with the maximum edges-to-vertices ratio d^*, is solvable in polynomial time. We discuss algorithms for this problem and the computation of a graph orientation with the lowest maximum indegree, which is equal to $\lceil d^* \rceil$. This value also equals the pseudoarboricity of the graph. We show that it can be computed in $\mathcal{O}(|E|^{3/2}\sqrt{\log\log d^*})$ time, and that better estimates can be given for graph classes where d^* satisfies certain asymptotic bounds. These runtimes are achieved by accelerating a binary search with an approximation scheme, and a runtime analysis of Dinitz's algorithm on flow networks where all arcs, except the source and sink arcs, have unit capacity. We experimentally compare implementations of various algorithms for the densest subgraph and pseudoarboricity problems. In flow-based algorithms, Dinitz's algorithm performs significantly better than push-relabel algorithms on all instances tested.

1 Introduction

1.1 Preliminaries and Problem Definition Let $G = (V, E)$ be a simple, finite graph with $V \neq \emptyset$. We will refer to this type of graphs as simple graphs or graphs throughout the paper. For asymptotic estimates we assume that $|E| \in \Omega(|V|)$, as isolated vertices are not of interest. The average density of a subgraph $H = (V_H, E_H) \subseteq G$ is defined as

$$d(H) := \frac{|E_H|}{|V_H|}.$$

An (induced) subgraph with maximum average density is called a densest subgraph of G. We will denote the maximum average density by $d^*(G)$ and the maximum vertex degree by $\Delta(G)$. We write d^* and Δ where the graph is clear from context.

The value $\lceil d^* \rceil$ has interesting connections to other concepts in combinatorics, as Theorem 1.1 shows. Arboricity (pseudoarboricity) is the minimum number of forests (pseudoforests) into which the set E can be decomposed. In a pseudoforest, every connected component is a pseudotree. A pseudotree of n vertices is connected and has at most n edges: it contains one cycle at most.

THEOREM 1.1. (PICARD AND QUEYRANNE [1]) *Let G be a simple graph. For the pseudoarboricity $P(G)$, we have $P(G) = \lceil d^*(G) \rceil$ and for the arboricity $\Gamma(G)$, we have $\Gamma(G) \in \{\lceil d^*(G) \rceil, \lceil d^*(G) \rceil + 1\}$.*

1.2 Related Work

The Densest Subgraph The first algorithm for the densest subgraph, based on network flow formulations, is due to Lawler [2, Chapter 4]. Picard and Queyranne [1] give an approach based on solving $|V|$ flow problems. Goldberg [3] solves the problem (and weighted generalizations) with a parameterized flow network. Up to $\log|V|$ maximum flow computations are performed on the original graph, which is augmented with source and sink arcs, in a binary search for the value d^*. With the Goldberg-Rao algorithm [4] for integral capacities (the value d^* is a rational number), it can be made to run in $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3}) \log(|V|^2/|E|) \log^2|V|)$.

The algorithm by Gallo et al. [5] solves parametric flow problems. The authors apply it to Goldberg's network. The total running time is bounded by $\mathcal{O}(|V||E| \log(|V|^2/|E|))$, but the algorithm does not generally benefit from improvements of maximum flow algorithms.

Chen [6] solves the problem (and a weighted generalization) as a flow problem on a parameterized bipartite graph. Cohen [7] formulates a linear program for determining d^*, which describes a flow problem on the same parameterized bipartite graph. Charikar [8] also formulates a linear program for determining d^* and addresses the density problem in directed graphs. Khuller and Saha [9] propose a flow-based algorithm for the directed case, which again involves a binary search.

Georgakopoulos and Politopoulos [10] generalize the problem to set-systems. For the case of simple graphs, their method is essentially equivalent to Goldberg's. However, they show that after an unsuccessful test, some vertices may be removed from the flow network. While the authors could not prove how this affects the runtime, they expect practical performance to be faster by a factor of about $\Theta(\log|V|)$. They also propose a parameterized linear program for weighted set-systems.

*Institute of Computer Science, Johannes Gutenberg University Mainz, Germany (mablumen@uni-mainz.de).

Several researchers describe a greedy algorithm that returns a 2-approximation for d^* in $\mathcal{O}(|V|+|E|)$ [11, 12, 13, 14, 8, 15, 10]. Georgakopoulos and Politopoulos [10] describe a generalization to set-systems, Khuller and Saha [9] address the case of directed graphs.

The densest subgraph problem has several applications, see e.g. [10, 16, 17].

Pseudoarboricity and Arboricity Determining arboricity and pseudoarboricity are special cases of the covering problem for matroid sums. The first matroid partitioning algorithm is due to Edmonds [18]. Gabow and Westermann [19, 20] propose algorithms specifically for arboricity and pseudoarboricity, the one for the latter runs in $\mathcal{O}(|E| \min(\sqrt{|E| \log |V|}, (|V| \log |V|)^{2/3}))$ time, being the currently fastest known algorithm. Gabow [21] shows that the arboricity of a graph can be computed in $\mathcal{O}(|E|^{3/2} \log(|V|^2/|E|))$.

Venkateswaran [22] proposes an $\mathcal{O}(|E|^2)$ algorithm for finding an orientation of the graph where the maximum indegree is lowest. This number is equal to $\lceil d^* \rceil$ and thus to pseudoarboricity [23, 13, 24]. Kowalik [24] presents an approximation scheme for $\lceil d^* \rceil$ which runs in $\mathcal{O}\left(|E| \log |V| \max\{1, \log d^*\}/\epsilon\right)$ time and returns an orientation where the maximum indegree is at most $\lceil (1+\epsilon) d^* \rceil$ for $\epsilon > 0$. Again, it involves a binary search and a parameterized flow problem on a network with integral capacities, which is solved using Dinitz's algorithm. The key idea is that the length of the shortest augmenting path can be bounded by a function that involves the desired approximation quality.

Asahiro et al. [25] propose a $(2 - 1/\lceil d^* \rceil)$-approximation algorithm for weighted graphs which runs in $\mathcal{O}(|E|^2)$ time. They also propose an exact flow-based algorithm for the unweighted case which runs in $\mathcal{O}(|E|^{3/2} \log d^*)$ time. They use the analysis of Even and Tarjan for unit capacity networks [26]. Essentially the same algorithm is proposed by Bezáková in unpublished work [15], and Aichholzer et al. [13] claim the same runtime on a parameterized bipartite flow network, without giving details.

Arboricity is a frequent measure of graph density. Low indegree (or outdegree) orientations have several applications. Refer to [24] for an overview.

1.3 Contributions

- We generalize the runtime analysis of Dinitz's algorithm for unit capacity networks [26] to 'almost unit capacity networks'. This analysis is applied to Goldberg's method for an improved runtime bound (Corollary 3.1).
- We show new runtime bounds for the pseudoarboricity problem (Theorem 1.2). Slightly better bounds may be found in certain cases, these are omitted for clarity. All bounds but one are achieved by accelerating a binary search with an approximation scheme (Section 3.3). The remaining bound is achieved with Gabow's algorithm for arboricity after a fast preprocessing of the graph (Section 6). We also discuss the application of a recent maximum flow algorithm by Mądry [27] in Section 4.
- We implement several algorithms mentioned in Section 1.2 for the density and pseudoarboricity problems, and compare them in a benchmark. To the best of our knowledge, this is the first experimental comparison performed for this problem.

THEOREM 1.2. *Pseudoarboricity and a corresponding partition into pseudoforests can be computed in the runtime bounds stated in Table 1.*

2 Bounds for Graph Density

Bounds can be of theoretical and practical value, e.g. in a binary search. A trivial lower bound for d^* is $|E|/|V|$ (by considering the whole graph as a subgraph), and a trivial upper bound is $|E|$. In the following, we prove better upper bounds, which we will refer to later.

LEMMA 2.1. *For a simple graph $G = (V, E)$ and a densest subgraph $H = (V_H, E_H) \subseteq G$, we have*

$$|V_H| \geq \sqrt{2|E_H| + \frac{1}{4}} + \frac{1}{2}.$$

Proof. If $|E| = 0$, the claim is immediate. If $|E| \geq 1$, any densest subgraph must contain at least two vertices. A subgraph with $|V_H|$ vertices cannot have more edges than the complete graph on $|V_H|$ vertices, thus we have $|E_H| \leq (|V_H|^2 - |V_H|)/2$. The claim follows by solving for $|V_H|$. □

PROPOSITION 2.1. *For a simple graph $G = (V, E)$, we have*

$$d^*(G) \leq \frac{1}{4}\left(\sqrt{8|E|+1} - 1\right).$$

Proof. For a densest subgraph (V_H, E_H), we have by Lemma 2.1

$$d^* = \frac{|E_H|}{|V_H|} \leq \frac{|E_H|}{\sqrt{2|E_H| + \frac{1}{4}} + \frac{1}{2}} = \frac{1}{4}\left(\sqrt{8|E_H|+1} - 1\right) \leq \frac{1}{4}\left(\sqrt{8|E|+1} - 1\right),$$

where we use the fact that $x/(\sqrt{2x + 1/4} + 1/2) = (\sqrt{8x+1} - 1)/4$ for all $x \geq 0$, which can be easily verified. □

Table 1: New runtime bounds for the pseudoarboricity problem, depending on d^*. Here, $\log^*$ denotes the iterated logarithm in base 2.

Bound for d^*	Runtime	Claim No.
$\Omega\left(\sqrt{\lvert E\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\right)$	(0)
$-$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\sqrt{\log\log d^*}\right)$	(I)
$\mathcal{O}\left(\frac{\sqrt{\lvert E\rvert}}{\log\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\log^* d^*\right)$	(II)
$\mathcal{O}\left(\frac{\sqrt{\lvert E\rvert}}{\log^2\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\right)$	(III)
$\mathcal{O}\left(\frac{(\lvert V\rvert\log\log\lvert V\rvert)^{2/3}}{\log\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert(\lvert V\rvert\log\log d^*)^{2/3}\right)$	(IV)
$\mathcal{O}\left(\frac{\lvert V\rvert^{2/3}}{\log\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert\lvert V\rvert^{2/3}\log^* d^*\right)$	(V)
$\mathcal{O}\left(\frac{\lvert V\rvert^{2/3}}{\log^2\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert\lvert V\rvert^{2/3}\right)$	(VI)

The bound from Proposition 2.1 is better than the arboricity bound

$$\Gamma \leq \left\lceil \sqrt{|E|/2 + |V|/4} \right\rceil \tag{2.1}$$

by Chiba and Nishizeki [28, Lemma 1], which is also a bound for pseudoarboricity by Theorem 1.1. The bound $\Gamma \leq \lceil |V|/2 \rceil$, which the authors derive from (2.1), can be improved to $d^* \leq \Delta(G)/2$.

PROPOSITION 2.2. *For a simple graph $G = (V, E)$, we have $d^*(G) \leq \Delta(G)/2$.*

Proof. Consider a densest subgraph $H = (V_H, E_H)$ of G. Let $\deg_H(v)$ denote the number of vertices in V_H adjacent to $v \in V_H$.

Assume that $d^* = \frac{|E_H|}{|V_H|} > \frac{\Delta}{2}$. We obtain

$$\begin{aligned} |E_H| > \frac{|V_H|\,\Delta}{2} &\geq \frac{\sum_{v\in V_H}\deg(v)}{2} \\ &\geq \frac{\sum_{v\in V_H}\deg_H(v)}{2} = |E_H|, \end{aligned}$$

a contradiction. □

An alternative proof of Proposition 2.2 utilizes the LP given by Cohen [7]. It can be found in Appendix A.

Gabow and Westermann [20, Lemma 4.1] provided, by considering matroid sums, bounds for arboricity and pseudoarboricity, the latter being

$$P(G) \leq \left\lfloor \frac{1}{4}\left(\sqrt{8|E| - 7} + 3\right)\right\rfloor$$

for $|E| \neq 0$. This bound is slightly tighter than the bound in Proposition 2.1 after adding one to account for the ceiling function in $P(G) = \lceil d^*(G) \rceil$. However, the methods employed in our proof are more elementary and the bound is tight for density as it holds with equality for complete graphs.

3 Almost Unit Capacity Networks

3.1 Preliminaries and Capacity Reduction We assume familiarity with flow problems. We will call vertices of a flow network *nodes* and directed edges *arcs* to avoid confusion. However, these often correspond to vertices and edges of the graph we are concerned with, so V and E will be used in the analysis.

DEFINITION 3.1. (AUC, AUC-2) *Let $G = (V, E)$ be a directed graph and let $N = (V \dot\cup \{s, t\}, E \cup E_s \cup E_t, c)$ be a flow network ("G-network") with*

$$E_s \subseteq \{s\} \times V,\ E_t \subseteq V \times \{t\},\ c : E \cup E_s \cup E_t \longrightarrow \mathbb{N}.$$

N is called an almost unit capacity (AUC) G-network if $c(u, v) \leq 1$ for all $(u, v) \in E$. If merely $c(u, v) \leq 2$ holds, N is called an AUC-2 G-network.

We consider nonexistent arcs to have zero capacity and assume that no parallel arcs are present since they can be merged into a single arc. The definition of AUC-2 networks will be needed in the following section as residual networks of AUC networks can have capacities of two per arc. We now prepare the reduction of large source and sink arc capacities.

LEMMA 3.1. *Let N be a flow network. There exists a maximum flow f in N with*

$$f(s, v), f(v, t) \geq F_v := \min(c(s, v), c(v, t))$$

for all $v \in V$. Let N' denote a copy of N with capacities $c'(s, v), c'(v, t)$ reduced by F_v for $v \in V$. A maximum flow in N' plus the flow F_v on the paths $s \to v \to t$ is one such flow f.

Intuitively, this means that one should send as much flow as possible on the length-2 paths $s \to v \to t$. The proof can be found in Appendix B.

PROPOSITION 3.1. *Let N be an AUC-2 G-network for a directed graph $G = (V, E)$, where source and sink arc capacities are bounded by a polynomial in $|V|$. The maximum flow can be found by running a linear-time preprocessing algorithm and invoking any maximum flow algorithm on the obtained AUC-2 G-network $\tilde{N}$ with bounded total source and sink arc capacities $C_{s,t}(\tilde{N}) \leq 2|E|$ and bounded maximum flow $M(\tilde{N}) \leq |E|$.*

Proof. We subtract the value F_v as defined in Lemma 3.1 from $c(s, v), c(v, t)$ for every $v \in V$ to obtain c' (with other capacities adopted without change).

Since for every v, at least one of these two arcs now has zero capacity, every flow-carrying path $s \rightsquigarrow v \rightsquigarrow t$ must pass through vertices in $V \setminus \{v\}$. Thus, the flow $s \to v$ is bounded by $\text{outdeg}(v)$ and the flow $v \to t$ is bounded by $\text{indeg}(v)$.

For any vertex with $c'(s, v) > \text{outdeg}(v)$ we can now safely set $\tilde{c}(s, v) = \text{outdeg}(v)$ for all $v \in V$. Likewise, for v with $c'(v, t) > \text{indeg}(v)$ we can set $\tilde{c}(v, t) = \text{indeg}(v)$ for all $v \in V$. Other capacities are adopted. Call this AUC-2 G-network $\tilde{N}$. Its total source and sink arc capacities are

$$\begin{aligned} C_{s,t}(\tilde{N}) &= \sum_{v \in V} (\tilde{c}(s, v) + \tilde{c}(v, t)) \\ &\leq \sum_{v \in V} (\text{outdeg}(v) + \text{indeg}(v)) = 2|E|. \end{aligned}$$

Moreover, the maximum flow is bounded by $|E|$ since the cut $(\{s\}, V \cup \{t\})$ has a capacity of $|E|$ at most. The capacity reduction takes $\mathcal{O}(|V| + |E|)$ time. □

For simple graphs, where $|E|$ is counted in the undirected way, we obtain the bound $M(\tilde{N}) \leq 2|E|$ for AUC-2 networks. For AUC networks, this bound can be improved to $M(\tilde{N}) \leq |E|$ by considering the cut $(S, \bar{S})$ where $S = \{s\} \cup \{v \in V \mid \tilde{c}(v, t) = 0\}$.

We apply these results to an example. For a simple graph G, Goldberg's flow network [3] is an example of a parameterized AUC G-network:

$$\begin{aligned} c(s, v) &= |E| \quad \forall v \in V \\ c(u, v) = 1 = c(v, u) &\quad \forall uv \in E \\ c(v, t) &= |E| + 2g - \deg(v) \quad \forall v \in V. \end{aligned}$$

For some guess $g \geq 0$ for d^*, the network has $(\{s\}, V \cup \{t\})$ as a minimum cut if and only if $g \geq d^*$. By applying Lemma 3.1 and Proposition 3.1, we obtain a flow network where the maximum flow value is bounded by $|E|$ instead of $|V||E|$.

3.2 Dinitz's Algorithm on AUC Networks We now investigate how Dinitz's algorithm [29] performs on AUC-2 G-networks. Assume that G is simple. Recall that Dinitz's algorithm works in phases. In each phase, a blocking flow is found. First, we give a generalization of a proposition by Kowalik [24, Proposition 3], the proof can be found in Appendix B.

PROPOSITION 3.2. *Each phase of Dinitz's algorithm runs in $\mathcal{O}(|E|)$ on an AUC-2 G-network N with bounded total source and sink arc capacities $C_{s,t}(N) \leq 2|E|$.*

We generalize theorems of Even and Tarjan for unit-capacity networks [26] to determine runtime bounds for AUC-2 networks with bounded maximum flow. The proofs are analogous. They are presented in Appendix B. A key fact we exploit is that for the maximum flow value M, we have $M \leq 2|E| < |V|^2$ and thus $\sqrt{M} < |V|$.

LEMMA 3.2. *Let N be an AUC-2 G-network with bounded maximum flow value $0 \neq M \leq 2|E|$. For the zero flow, the (unit) distance l from s to t is at most*

$$l \leq \min\left(\frac{6|E|}{M}, \frac{\left(1 + \sqrt{8}\right)|V|}{\sqrt{M}}\right).$$

THEOREM 3.1. *Dinitz's algorithm runs in $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3}))$ time on an AUC-2 G-network with bounded total source and sink arc capacities $C_{s,t}(N) \leq 2|E|$.*

3.3 Faster Algorithms for Pseudoarboricity We now demonstrate how an approximation algorithm can be used to narrow a search interval for a binary search (as in Goldberg's method, for example). Note that Bezáková [15] has suggested such a technique before.

LEMMA 3.3. *Let I be an interval of integers, and let $x \in I$ be the minimum value for which some property holds. Furthermore, let the property hold for all $x' \in I$ with $x' \geq x$. Given an approximation $d \in I$ for x with $x \leq d \leq \lceil (1 + \delta)x \rceil$ for some $\delta > 0$, a binary search for x can be realized with $\mathcal{O}(\log(\delta x))$ tests.*

Proof. We have $d - 1 \leq (1 + \delta)x$ and obtain the lower bound $(d - 1)/(1 + \delta) \leq x$. The interval $I \cap \{\lceil (d - 1)/(1 + \delta) \rceil, ..., d - 1\}$ of integral values for x that remain to be checked can be shown to have length $L < 2\delta x + 2$. See Appendix C for the calculations. □

The greedy approximation algorithm (see Section 1.2) for d^* outputs an integral value d_G satisfying $d^* \leq d_G \leq 2d^*$. It runs in $\mathcal{O}(|V| + |E|)$ time.

COROLLARY 3.1. *To compute $\lceil d^* \rceil$, Goldberg's method can be made to run in*

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|}, |V|^{2/3}\right) \log d^*\right).$$

Proof. Obtain the 2-approximation d_G with the greedy algorithm. Since d_G is integral, we have $\lceil d^* \rceil \leq d_G \leq 2d^* \leq 2\lceil d^* \rceil = \lceil 2\lceil d^* \rceil \rceil$. By applying Lemma 3.3 with $x = \lceil d^* \rceil$, $d = d_G$ and $\delta = 1$, performing the binary search is possible with $\mathcal{O}(\log d^*)$ tests (instead of $\log |V|$). For every integral guess g, we construct Goldberg's network and reduce capacities by Proposition 3.1. Then, Theorem 3.1 can be applied for each test. □

Corollary 3.1 is an improvement over previous exact algorithms based on network flow, or their runtime estimates (see Section 1.2).

Similarly to Goldberg's method, the value $\lceil d^* \rceil$ can be found in a binary search on a parameterized 're-orientation' flow network [24]. This method utilizes a theorem of Frank and Gyárfás [23, Theorem 1]. The theorem states that there exists an orientation of a simple graph such that the outdegree (or equivalently, indegree) of every vertex is bounded by $\lceil d^* \rceil$, and this value is minimal.

An intuitive explanation of the re-orientation flow network is as follows: For some tentative integer d, we try to re-orient an arbitrary given orientation $\vec{G}$ such that every vertex has at most d ingoing edges (this is called a d-orientation). To do so, we add a source with arcs to every vertex which has more than d ingoing edges: this vertex has to flip at least $\text{indeg}(v) - d$ ingoing edges (more, if outgoing edges flip as well) to reach level d. Likewise, we add a sink with arcs from every vertex with $\text{indeg}(v) < d$ to it. These vertices may rise up to level d by flipping edges.

Formally, we introduce source and sink nodes $s, t \notin V$, and define the set of arcs A on $V \cup \{s, t\}$ as follows:

$$\begin{aligned} (s,v) \in A:\ & c(s,v) = \text{indeg}_{\vec{G}}(v) - d \Leftrightarrow \text{indeg}_{\vec{G}}(v) > d, \\ (v,t) \in A:\ & c(v,t) = d - \text{indeg}_{\vec{G}}(v) \Leftrightarrow \text{indeg}_{\vec{G}}(v) < d, \\ (u,v) \in A:\ & c(u,v) = 1 \qquad\qquad\quad \Leftrightarrow u \leftarrow v \text{ in } \vec{G}. \end{aligned}$$

Notice that this network is an AUC network for a directed graph. Thus an analogue of Corollary 3.1 can be proved. As described in its proof, the greedy algorithm can be used to narrow the length of the search interval such that $\mathcal{O}(\log d^*)$ tests suffice. Thus, there is no need to perform an exponential search as described by Kowalik [24] to find an initial 2-approximation. The greedy algorithm also computes a d_G-orientation, however, any orientation can be used for re-orientation.

The following lemma was proved by Kowalik and is a stronger variant of a lemma by Brodal and Fagerberg [30, Lemma 2]. It can be further generalized to Lemma C.1, which can be applied to the densest subgraph problem.

LEMMA 3.4. ([24, LEMMA 2]) *Let $\vec{G}$ be a d-orientation of a graph G, and let $d > d^*(G)$. Then for every vertex $v \in V$, the unit distance in $\vec{G}$ to a vertex with indegree smaller than d does not exceed $\log_{d/d^*} |V|$.*

Kowalik uses this fact to stop a test with Dinitz's algorithm after at most $\lfloor 2 + \log_{1+\epsilon} |V| \rfloor$ phases, thereby turning the re-orientation algorithm into an $(1+\epsilon)$-approximation scheme for $\lceil d^* \rceil$. This is justified since if $d \geq \lceil (1+\epsilon) d^* \rceil$, we have $d/d^* \geq (1+\epsilon)$, and the length of the shortest augmenting path increases with every phase. Otherwise, the initial orientation happened to be a $(1+\epsilon)$-approximation.

Finding an integer d such that $\lceil d^* \rceil \leq d \leq \lceil (1+\epsilon) d^* \rceil$ is now possible in $\mathcal{O}(|E| \log(|V|)/\epsilon \log d^*)$ time by a Taylor expansion of $\ln(1+\epsilon)$ for $\epsilon > 0$.

Note that Kowalik's scheme never stops the binary search early as it does not detect when the error ϵ is undershot; it solely relies on the stopping criterion derived from Lemma 3.4. Thus, if this criterion is never met during the execution, Kowalik's scheme outputs the optimum solution.

To compute $\lceil d^* \rceil$ exactly in general, one could try to set $\epsilon = 1/d^*$, as the algorithm then outputs a d-orientation where $d \leq \lceil d^* + 1 \rceil = \lceil d^* \rceil + 1$. A final test with Dinitz's algorithm for $d - 1$ would allow us to determine the exact value. The runtime is bounded by $\mathcal{O}(|E| \log(|V|) d^* \log d^* + |E| \min(\sqrt{|E|}, |V|^{2/3}))$. Since we do not know d^*, we can use the approximation $d^* \leq d_G \leq 2d^*$ of the greedy algorithm to set a slightly smaller $\epsilon = 1/d_G$. Of course, we can use Theorem 3.1 for the approximation phase as well if d^* is large to get a better runtime bound. Still, this method is in general not better than the Gabow-Westermann algorithm [20], which runs in $\mathcal{O}(|E| \min(\sqrt{|E| \log d^*}, (|V| \log d^*)^{2/3}))$ time if the greedy approximation algorithm is used beforehand. Interestingly, the Gabow-Westermann algorithm needs asymptotically less time than $\log d^*$ worst-case Dinitz executions on unit capacity networks. Let us explore this notion.

We divide the re-orientation algorithm into two (or more) phases and balance their runtimes. For the first phase, we start the approximation scheme with some $\epsilon > 0$. It returns a d-orientation where $d \leq \lceil (1+\epsilon) d^* \rceil$. In the second phase, we continue the binary search with Dinitz's algorithm on the parameterized flow network, but use our results: We need $\mathcal{O}(\log(\epsilon d^*))$ tests on the narrowed interval by Lemma 3.3, and can use a different analysis (Theorem 3.1), which is beneficial if d^* is large. For the second phase, a different algorithm could be used (e.g., a different flow algorithm or an entirely different method such as Goldberg's). We now prove six of seven bounds for the main theorem.

Proof (Theorem 1.2, claims (I)-(VI) in Table 1). For all claims, we initially compute the 2-approximation

d_G greedily in linear time. The values d_G and $d_G/2$ will be used to set parameters. However, we will suggestively call these values d^* as this does not change the runtime estimates asymptotically and simplifies the presentation. We use $\simeq$ to signify the constant factors when setting parameters. We further assume that numeric computations are performed to arbitrary precision, and that $d^* > 1$.

Consider claim (I) in Table 1. Check whether $d^* \leq |V|^{0.49}$ (any fixed exponent less than 0.5 would do). If yes, see the proof of claim (III).

Otherwise, we have $d^* > |V|^{0.49}$ and therefore $\log d^* > 0.49 \log |V|$. We set $\epsilon \simeq \log(|V|)^2/d^*$ and thus the first phase runs in $\mathcal{O}(|E|d^*)$ time. By Proposition 2.1, this is always in $\mathcal{O}(|E|^{3/2})$.

A binary search on the narrowed search interval now needs $\mathcal{O}(\log(\epsilon d^*)) = \mathcal{O}(\log \log d^*)$ tests by Lemma 3.3. The re-orientation algorithm (or Goldberg's) can thus perform the second phase in time $\mathcal{O}(|E|^{3/2} \log \log d^*)$ by Theorem 3.1. The Gabow-Westermann algorithm can perform the second phase in $\mathcal{O}(|E|^{3/2}\sqrt{\log \log d^*})$ time by setting the parameter for the balanced binary search used in it appropriately (see [20, Section 4]). The claim follows.

For claims (II) and (V), assume that either $d^* \in \mathcal{O}(\sqrt{|E|}/\log |V|)$ or $d^* \in \mathcal{O}(|V|^{2/3}/\log |V|)$. We run the approximation scheme in $i = 1, ..., \log^* d^* - 1$ phases[1], each time on the search interval left after the previous phase, with parameters

$$\epsilon_1 \simeq \frac{\log d^*}{d^*}, \epsilon_2 \simeq \frac{\log \log d^*}{d^*}, \epsilon_3 \simeq \frac{\log \log \log d^*}{d^*}, ...$$

For convenience, define $\log^{\times i} a := \log \cdots \log a$ (i times) and $\log^{\times 0} a := a$. We prove inductively that for the interval I_i leftover after phase i, we have $|I_i| \in \mathcal{O}(\log^{\times i} d^*)$ and thus phase i runs in $\mathcal{O}(|E|^{3/2})$ resp. $\mathcal{O}(|E||V|^{2/3})$ time. The induction basis holds as for the initial search interval I_0, we have $|I_0| \in \mathcal{O}(d^*)$ with the greedy 2-approximation.

Let the induction hypothesis hold for $i-1$. In phase i, we perform Kowalik's scheme on I_{i-1} with ϵ_i in

$$\mathcal{O}\left(\frac{|E| \log |V| d^*}{\log^{\times i} d^*} \log \log^{\times (i-1)} d^*\right).$$

Thus the phase runs in $\mathcal{O}(|E|^{3/2})$ resp. $\mathcal{O}(|E||V|^{2/3})$ time and leaves an interval of length $|I_i| \in \mathcal{O}(\log |I_{i-1}|) = \mathcal{O}(\log^{\times i} d^*)$ by Lemma 3.3. It is important to ensure that the hidden constants do not accumulate during the iterated process. In the proof of Lemma 3.3, we saw that a multiplicative constant of two and an additive constant of two may be introduced when narrowing the interval. We can divide ϵ_i by a constant $c > 2$ to avoid accumulating ever larger constants. After $\log^* d^* - 1$ phases, we have narrowed the search interval to the length of $\mathcal{O}(\log^{\times \log^* d^*} d^*) = \mathcal{O}(1)$. The final phase consists of a constant number of tests and runs in $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3}))$ time with the re-orientation algorithm. Thus claims (II) and (V) are proven.

For claims (III) and (VI), we set $\epsilon \simeq 1/d^*$. The first phase clearly runs in time $\mathcal{O}(|E|^{3/2})$ or $\mathcal{O}(|E||V|^{2/3})$, respectively. The second phase is a single test with Dinitz's algorithm, which takes $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3}))$. Note that substituting one $\log |V|$ in the bounds on d^* with $\log d^*$ does not produce a stronger result.

For claim (IV), assume that $d^* \in \mathcal{O}((|V| \log \log d^*)^{2/3}/\log |V|)$. One can see from a case analysis that this is not more restrictive than the bound in the table. Choose $\epsilon \simeq \log(d^*)/d^*$. The first phase runs in time $\mathcal{O}(|E| \log |V| d^*) \subseteq \mathcal{O}(|E|(|V| \log \log d^*)^{2/3})$. The second phase runs in time $\mathcal{O}(|E||V|^{2/3} \log \log d^*)$ with the re-orientation algorithm and in $\mathcal{O}(|E|(|V| \log \log d^*)^{2/3})$ with the Gabow-Westermann algorithm.

Once a $\lceil d^* \rceil$-orientation has been computed, it can be converted into the optimal pseudoforest partition in linear time [24, Proposition 1]. If $\lceil d^* \rceil$ is known, it is possible to compute a $\lceil d^* \rceil$-orientation in $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3}))$ time. This runtime is also possible with matroid techniques [20, Theorem 3.2]. □

4 Utilizing Mądry's Algorithm

Recently, it was shown by Mądry [27] that the maximum flow problem on unit capacity networks can be solved in $\tilde{\mathcal{O}}(|E|^{10/7})$ time. This improves the long-standing time bound by Even and Tarjan [26]. Mądry's definition of flow networks allows parallel arcs. We can thus convert AUC networks into unit capacity networks with parallel arcs.

THEOREM 4.1. *The pseudoarboricity of a simple graph $G = (V, E)$ can be determined in $\tilde{\mathcal{O}}(|E|^{10/7})$ time with Mądry's flow algorithm.*

Proof. The proof uses Goldberg's or the re-orientation flow network. For Goldberg's AUC G-network N, we apply Proposition 3.1 to obtain a flow network N' where the total source and sink arc capacities are bounded by $\mathcal{O}(|E|)$. We replace the source and sink arcs in N' by parallel unit capacity arcs, i.e. for $v \in V$, we replace the arc $s \to v$ by $c'(s, v)$ parallel arcs of capacity one, and the arc $v \to t$ by $c'(v, t)$ parallel arcs of capacity one. Call this flow network N^p. The number of unit capacity arcs in N^p is in $\mathcal{O}(|E|)$, so the network size

[1] Here, $\log^*$ denotes the iterated logarithm in base two.

increases only by a constant factor. A binary search method with Mądry's flow algorithm runs in $\tilde{\mathcal{O}}(|E|^{10/7})$ time. □

While this improves upon the bounds (0)-(III) in Table 1, we can also speed up the binary search with Kowalik's approximation scheme when using Mądry's algorithm if d^* satisfies certain bounds. Note that the bounds (IV)-(VI) in Table 1 are better if the graph is quite dense.

5 Linear Programs

Charikar [8], Georgakopoulos and Politopoulos [10], and Cohen [7] propose different linear programming formulations to compute d^*. Cohen's LP is a relaxation of the lowest maximum indegree orientation problem: each edge is fractionally oriented to its two end vertices. Formally, the LP is given as:

$$\begin{aligned} &\text{minimize} && d && \\ &\text{s.t.} && d \geq \sum_{uv \in E} f_{uv,v} && \forall v \in V \quad (5.2)\\ && && f_{uv,u} + f_{uv,v} = 1 && \forall uv \in E \quad (5.3)\\ && && f_{uv,u}, f_{uv,v} \geq 0 && \forall uv \in E \\ && && d \geq 0. \end{aligned}$$

It is easy to see that Constraint (5.3) allows us to reduce the number of variables from $2|E|+1$ to $|E|+1$, since we can substitute $f_{uv,u} = 1 - f_{uv,v}$. This is used in our LP implementation in Section 7.

It can be shown that a solution to Cohen's LP is a maximum flow in Goldberg's flow problem, where the guess parameter equals d in the LP solution (Proposition D.1). Furthermore, Cohen's LP is a proper subset of the dual of Charikar's LP, where Constraint (5.3) is $f_{uv,u} + f_{uv,v} \geq 1$. Note that Charikar's LP has also been investigated by Balalau et al. [17].

6 Preprocessing: Theory and Practice

In the densest subgraph problem, a vertex can be safely removed from the graph if its degree is smaller than d^*.

LEMMA 6.1. (KHULLER AND SAHA [9, PAGE 6]) *Let G be a simple graph. A vertex v with $\deg(v) < d^*(G)$ cannot be in a densest subgraph.*

Proof. Assume a vertex v with $\deg(v) < d^*$ is contained in a densest subgraph (V_H, E_H). Then $G' = G[V_H \setminus \{v\}]$ would have a density higher than d^* (the calculations are omitted for brevity). This is a contradiction since G' itself is a subgraph of G. □

We can repeatedly remove vertices with Lemma 6.1 without changing the maximum density of the graph.

LEMMA 6.2. *Let $d \leq d^*(G)$ for a simple graph $G = (V, E)$. Then we can obtain a subgraph $G' \subseteq G$ with $d^*(G') = d^*(G)$ where $\deg_{G'}(v) \geq d$ for all $v \in V$ in $\mathcal{O}(|E|)$ time.*

Proof. We initialize a queue with all vertices whose degree is less than d, and store the current vertex degrees in an array. While the queue is not empty, a vertex u is dequeued. Mark it as removed and decrease the degrees of all its unmarked neighbors by one. If a neighbor's degree falls below d for the first time, add it to the queue. Once the queue is empty, build the graph induced by the set of unmarked vertices. The algorithm's correctness is obvious from Lemma 6.1. It runs in $\mathcal{O}(|E|)$ time. □

With an approximation algorithm, we can do a fast preprocessing with Lemma 6.2, which can significantly reduce the input size. The preprocessed graph is also possibly more dense. Gabow's algorithm for arboricity [21] runs in $\mathcal{O}(|E|^{3/2}\log(|V|^2/|E|))$ time, i.e., the logarithmic term becomes constant for dense graphs. We are able to use this to obtain a new runtime estimate.

LEMMA 6.3. *Let $G = (V, E)$ be a simple graph. Then a subgraph $G' = (V', E') \subseteq G$ with $d^*(G') = d^*(G) =: d^*$ and $|E'| \geq |V'|\, d^*/4$ can be obtained in $\mathcal{O}(|E|)$ time.*

Proof. Compute the 2-approximation $d^* \leq d_G \leq 2d^*$ in time $\mathcal{O}(|E|)$. Set $d := d_G/2$. We have $d \leq d^*$. Obtain the subgraph $G' = (V', E') \subseteq G$ from Lemma 6.2 with d in time $\mathcal{O}(|E|)$. For every vertex $v \in V'$, $\deg_{G'}(v) \geq d \geq d^*/2$. We have

$$2|E'| = \sum_{v \in V'} \deg_{G'}(v) \geq |V'|\, d^*/2,$$

thus $|E'| \geq |V'|\, d^*/4$. □

We are now able to prove that pseudoarboricity can be determined in $\mathcal{O}(|E|^{3/2})$ time if $d^* \in \Omega(\sqrt{|E|})$.

Proof (Theorem 1.2, claim (0) in Table 1). If we have $d^* \in \Omega(\sqrt{|E|})$, then $d^* \geq c\sqrt{|E|}$ for some $c > 0$. Obtain $G' = (V', E')$ from Lemma 6.3, we have

$$\begin{aligned} &|E'| \geq |V'|\, d^*/4 \geq |V'|\sqrt{|E|}\, c/4 \geq |V'|\sqrt{|E'|}\, c/4 \\ \Rightarrow\ &|E'| \geq |V'|^2\, c^2/16, \end{aligned}$$

and thus $|E'| \in \Omega(|V'|^2)$. The arboricity $\Gamma(G')$ can be computed in $\mathcal{O}(|E|^{3/2})$ time with Gabow's algorithm. By applying Theorem 1.1 we obtain

$$\lceil d^*(G) \rceil + 1 \geq \Gamma(G) \geq \Gamma(G') \geq \lceil d^*(G') \rceil = \lceil d^*(G) \rceil .$$

Therefore, a single test for $\Gamma(G') - 1$ suffices to determine pseudoarboricity (and a corresponding pseudoforest partition) in $\mathcal{O}(|E|^{3/2})$ time. See Section 3.3 for details. □

7 Experiments

We are not aware of performance comparisons of algorithms for the densest subgraph and pseudoarboricity problems. However, extraction of densest subgraphs is a common application. To this end, Tsourakakis et al. [16] implemented Goldberg's method and the greedy algorithm. Balalau et al. [17] use Charikar's linear program and the greedy algorithm to find minimal densest subgraphs with small overlap. The authors consider Goldberg's method unsuitable for their purposes due to poor performance; they did not elaborate which flow algorithm was used.

7.1 Tested Algorithms and Benchmark Setting

We tested binary search methods with maximum flow algorithms, which were implemented in Java 7 Update 79. We implemented Dinitz's (D) algorithm and variants of the push-relabel algorithm. The relabel-to-front variant (RF) runs in $\mathcal{O}(|V|^3)$ [31], the highest-label variant (HL) runs in $\mathcal{O}(|V|^2\sqrt{|E|})$ time [32]. For the highest label variant, we added the global relabeling and gap heuristics [33]. We also extended the highest-label variant to an algorithm for parametric flow networks as described Gallo et al. (P-HL) [5], which can be shown to perform at most $\mathcal{O}(|V|^2\sqrt{|E|})$ nonsaturating pushes in total [34, Theorem 2.4].

These algorithms are used to determine $\lceil d^* \rceil$ with the methods of Goldberg, Georgakopoulos and Politopoulos, and the re-orientation algorithm from Section 3.3. The latter performs exact computation without approximation phases. The reason for this is that the stopping criterion was not met for the parameter choices in the proof of Theorem 1.2, and thus the approximation scheme is identical to the exact method. We give more details on the length of augmenting paths in Section 7.3.

We also implemented the greedy 2-approximation algorithm, Cohen's LP, and the parameterized Georgakopoulos-Politopoulos LP [10] with a binary search. We used Gurobi 6.0 [35], which is free for academic purposes, to solve the LPs. We report results for the dual simplex method, which was consistently faster and more memory-efficient than the primal simplex method.

The tests were run on a single core of an Intel i7-5820K CPU with 3.3 GHz and 64 GB DDR-4 RAM. The operating system was Ubuntu 14.04 (64-bit). The algorithms were started with the bounds from Section 2. All tests were repeated with the preprocessing from Lemma 6.3 to reduce the input sizes.

7.2 Input Graphs

We used large simple graphs from the Stanford SNAP database [36] as input graphs, namely the *Amazon*, *DBLP*, *YouTube*, *LiveJournal* and *Orkut* networks with about 900K, 1M, 3M, 35M and 117M edges, respectively.

Even and Tarjan [26] propose a family of flow networks on which Dinitz's algorithm needs as much time as its worst-case runtime estimate. Since our flow networks do not belong to this family, we propose a graph family where we expect the shortest augmenting paths in the corresponding G-networks to become very long because of a mixture of strongly varying degrees, local densities, and a large diameter. For $n \in \mathbb{N}$, we define graph G_n as a collection of the complete graphs $K_1, K_2, ..., K_n$, where every vertex of K_i is additionally connected to all vertices of K_{i+1} for $i = 1, ..., n-1$. G_n has $n(n+1)/2$ vertices and $(n^3 - n)/2$ edges in total and an average density $|E|/|V| = n - 1 \in \Theta(\sqrt{|V|})$. If $n \geq 2$, the vertices in K_{n-1} have a degree of $3n-4$ in G_n, which is the maximum degree $\Delta(G_n)$. By Lemma 2.2, we have $d^* \leq 3n/2 - 2 \in \Theta(\sqrt{|V|})$, so $d^* \in \Theta(\sqrt{|V|})$. We denote the smallest $\epsilon > 0$ for which the stopping criterion of the approximation scheme is met by $\tilde{\epsilon}$. We expect that $\tilde{\epsilon}(G_n) \to 0$ as $n \to \infty$.

7.3 Results

The runtime results of the different exact algorithms are presented in Table 2. Flow-based methods with Dinitz's algorithm were fastest on all instances. Dinitz's algorithm was significantly faster than the push-relabel variants. We note that with push-relabel algorithms, the number of relabelings in the most expensive test in the binary search was sometimes more than a thousand times greater than in the fastest test. The highest-label variant performs better than both the relabel-to-front variant and the parametric extension.

The Georgakopoulos-Politopoulos method, which removes vertices after unsuccessful tests, is considerably faster than Goldberg's on a few instances, but slightly slower on others. In both the original and the preprocessed instances, an unsuccessful test removes 5-70% of the remaining vertices. However, there is no known estimate of how many vertices can be expected to be removed. It may also happen that all tests but one are successful; this occurs in the *Amazon* instance.

Cohen's linear program is solved considerably faster than the Georgakopoulos-Politopoulos LP with a binary search. A comparison between Cohen's LP and flow-based methods using the highest label variant is inconclusive.

The greedy 2-approximation algorithm needed less than 8 seconds on all instances, on half of them, it runs in less than a second. The preprocessing also needs merely seconds. It reduces the number of edges of the graphs from the SNAP database by $13 - 98\%$ (Table 3).

We report our findings on the augmenting path lengths in the execution of Kowalik's scheme in Table

Table 2: Time (in seconds, rounded) needed to compute the pseudoarboricity of ten different graphs with several algorithms. The suffix -P denotes instances reduced by preprocessing. The network flow algorithms were Dinitz's algorithm (D), the relabel-to-front (RF) and highest-label (HL) variants of the push-relabel algorithm, and the extension of the latter variant by Gallo et al., which directly solves parametric flow problems (P-HL). If the computation required more than the available 64 GB of RAM, 'out of memory' (*oom*) is stated in the table. Computations were stopped after 10 hours.

			Network Flow Methods									
	LPs		Goldberg's				Georgak.-Politop.			Re-orientation		
Graph	Cohen's	G.-Pol.	D	RF	HL	P-HL	D	RF	HL	D	RF	HL
Amazon	125	177	3	5459	200	9164	2	5580	262	23	1247	189
DBLP	7	34	2	4111	11	8417	0	273	7	4	401	6
Youtube	242	657	15	>10h	665	>10h	5	>10h	635	4	>10h	210
LiveJ.	299	21468	113	>10h	14047	>10h	34	>10h	2165	251	>10h	1264
Orkut	>10h	*oom*	391	>10h	>10h	>10h	119	>10h	18252	1012	>10h	>10h
G_{100}	23	569	0	114	1	44	0	160	1	0	92	1
G_{200}	631	31670	7	7925	22	1306	2	7399	29	6	3832	11
G_{400}	12857	*oom*	84	>10h	434	>10h	21	>10h	554	90	>10h	352
G_{600}	>10h	*oom*	427	>10h	2595	>10h	97	>10h	3146	469	>10h	1329
G_{800}	*oom*	*oom*	995	>10h	4870	>10h	228	>10h	4773	1508	>10h	5619
Amazon-P	75	105	1	1468	73	1011	2	2007	114	1	1528	100
DBLP-P	0	0	0	0	0	0	0	0	0	0	0	0
Youtube-P	33	50	1	495	4	63	0	464	3	0	930	4
LiveJ.-P	15	78	0	52	0	39	0	45	0	0	80	1
Orkut-P	33426	>10h	40	>10h	260	30964	24	>10h	176	33	>10h	144
G_{100}-P	19	313	0	53	0	45	0	51	0	0	52	0
G_{200}-P	638	19321	3	3335	6	1364	2	3167	6	1	3000	5
G_{400}-P	18712	*oom*	26	>10h	103	>10h	20	>10h	96	8	>10h	82
G_{600}-P	>10h	*oom*	117	>10h	627	>10h	86	>10h	605	28	>10h	548
G_{800}-P	*oom*	*oom*	284	>10h	2117	>10h	208	>10h	2073	80	>10h	4146

Table 3: Characteristics of the graphs tested. The value $\tilde{\epsilon}$ denotes the smallest value of ϵ for which the stopping criterion is met in Kowalik's approximation scheme (rounded to two decimal places), and k denotes the corresponding maximum pathlength. These values are empirical and may slightly depend on the initial orientation, as well as the implementation of Dinitz's algorithm.

		Original input				After Preprocessing			
Graph	$\lceil d^* \rceil$	Vertices	Edges	$\tilde{\epsilon}$	k	Vertices	Edges	$\tilde{\epsilon}$	k
Amazon	5	334,863	925,872	0.96	21	255,473	802,913	0.93	20
DBLP	57	317,080	1,049,866	67.2	5	280	13,609	5.55	4
YouTube	46	1,134,890	2,987,624	0.89	24	11,934	417,299	0.64	20
LiveJ.	194	3,997,962	34,681,189	2.54	12	3,128	539,742	0.86	14
Orkut	228	3,072,441	117,185,083	0.61	34	70,632	13,359,726	0.52	28
G_{100}	134	5,050	499,950	1.51	11	3,825	438,700	1.29	11
G_{200}	277	20,100	3,999,900	1.29	13	15,150	3,504,900	1.10	14
G_{400}	567	80,200	31,999,800	0.95	18	60,300	28,019,800	0.85	19
G_{600}	859	180,300	107,999,700	0.78	22	135,450	94,544,700	0.72	23
G_{800}	1152	320,400	255,999,600	0.70	25	240,600	224,079,600	0.65	26

3. On some graphs, the stopping criterion is not met for any $\epsilon < 1$, i.e. the exact solution is returned. (Choosing $\epsilon \geq 1$ is not reasonable because the linear-time 2-approximation algorithm is preferable.) On all input graphs, the choices for ϵ in the proof of Theorem 1.2 are smaller than the critical epsilon, i.e., the (first) approximation phase computes the optimum solution. This is easily recognized and the algorithm can be stopped.

8 Conclusion and Outlook

We presented a generalization of the analysis of unit capacity networks to 'almost unit capacity' networks. We used it to improve runtime bounds of the re-orientation, and, for integral guesses, Goldberg's methods. We also showed how Kowalik's approximation scheme can be used to obtain asymptotically faster algorithms for pseudoarboricity. However, this cannot be expected to significantly improve runtimes for graph sizes encountered in practice today. The recent development of 'electrical' flow algorithms such as Mądry's also gives rise to new runtime bounds. If these algorithms have practical relevance is a question which deserves attention in the future.

We implemented several algorithms for pseudoarboricity and compared their performance. The results show that Dinitz's algorithm provides a decisive advantage over push-relabel algorithms for this problem as the maximum length of the shortest augmenting paths is small. With Dinitz's algorithm, the flow-based methods outperform linear programs with a state-of-the-art solver. Preprocessing is an effective tool which drastically reduces runtimes on real-world input graphs. Theoretical estimates of how much certain graph families (e.g. graphs with power-law degree distributions) are reduced by preprocessing are another interesting research question.

Acknowledgements The author would like to thank the anonymous reviewers for their helpful comments, as well as Ernst Althaus for fruitful discussions.

References

[1] Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982.

[2] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[3] Andrew V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984.

[4] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, September 1998.

[5] Giorgio Gallo, Michael D. Grigoriadis, and Robert E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, February 1989.

[6] Y. L. Chen. A parametric maximum flow algorithm for bipartite graphs with applications. *European Journal of Operational Research*, 80(1):226–235, 1995.

[7] Nathann Cohen. Several graph problems and their LP formulation (explanatory supplement for the Sage graph library). `http://hal.inria.fr/inria-00504914`, July 2010.

[8] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '00, pages 84–95, London, UK, 2000. Springer-Verlag.

[9] Samir Khuller and Barna Saha. On finding dense subgraphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 597–608. Springer, 2009.

[10] George F. Georgakopoulos and Kostas Politopoulos. MAX-DENSITY revisited: a generalization and a more efficient algorithm. *Comput. J.*, 50(3):348–356, 2007.

[11] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, 1994.

[12] Guy Kortsarz and David Peleg. Generating sparse 2-spanners. *J. Algorithms*, 17(2):222–236, September 1994.

[13] Oswin Aichholzer, Franz Aurenhammer, and Günter Rote. Optimal graph orientation with storage applications. SFB-Report F003-51, SFB 'Optimierung und Kontrolle', TU Graz, Austria, 1995.

[14] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. In Rolf Karlsson and Andrzej Lingas, editors, *Algorithm Theory — SWAT'96*, volume 1097 of *Lecture Notes in Computer Science*, pages 136–148. Springer Berlin Heidelberg, 1996.

[15] Ivona Bezáková. Compact representations of graphs and adjacency testing. Master's thesis, Comenius University, Bratislava, Slovakia, April 2000. `http://people.cs.uchicago.edu/~ivona/PAPERS/GraphRepr.ps`.

[16] Charalampos E. Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, ed-

itors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 104–112. ACM, 2013.

[17] Oana Denisa Balalau, Francesco Bonchi, T.-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. Finding subgraphs with maximum total density and limited overlap. In Xueqi Cheng, Hang Li, Evgeniy Gabrilovich, and Jie Tang, editors, *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM 2015, Shanghai, China, February 2-6, 2015*, pages 379–388. ACM, 2015.

[18] Jack Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards Sect. B*, 69B:67–72, 1965.

[19] Herbert H. Westermann. *Efficient Algorithms For Matroid Sums*. PhD thesis, University of Colorado Boulder, USA, 1988.

[20] Harold N. Gabow and Herbert H. Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica*, 7(1-6):465–497, 1992.

[21] Harold N. Gabow. Algorithms for graphic polymatroids and parametric $\bar{s}$-sets. *Journal of Algorithms*, 26(1):48–86, 1998.

[22] Venkat Venkateswaran. Minimizing maximum indegree. *Discrete Applied Mathematics*, 143(1–3):374 – 378, 2004.

[23] András Frank and András Gyárfás. How to orient the edges of a graph? In András Hajnal and Vera T. Sós, editors, *COMBINATORICS: 5th Hungarian Colloquium, Keszthely, June/July 1976, Proceedings*, number 2 in Colloquia Mathematica Societatis János Bolyai, pages 353–364. North Holland Publishing Company, 1978.

[24] Łukasz Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In Tetsuo Asano, editor, *Algorithms and Computation*, volume 4288 of *Lecture Notes in Computer Science*, pages 557–566. Springer Berlin Heidelberg, 2006.

[25] Yuichi Asahiro, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. *International Journal of Foundations of Computer Science*, 18(02):197–215, 2007.

[26] Shimon Even and Robert E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, 1975.

[27] Aleksander Mądry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262. IEEE Computer Society, 2013.

[28] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, February 1985.

[29] E. A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation (translation from Russian). *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[30] Gerth S. Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, WADS '99, pages 342–351, London, UK, 1999. Springer-Verlag.

[31] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[32] J. Cheriyan and S.N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. In Kesav V. Nori and Sanjeev Kumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 338 of *Lecture Notes in Computer Science*, pages 30–48. Springer Berlin Heidelberg, 1988.

[33] Boris V. Cherkassky and Andrew V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[34] Dan Gusfield and Éva Tardos. A faster parametric minimum-cut algorithm. *Algorithmica*, 11(3):278–290, 1994.

[35] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[36] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`, June 2014.

A Bounds for Graph Density

Cohen's LP (see Section 5) can be used to prove Proposition 2.2.

Proof (Proposition 2.2, alternative). Set $f_{uv,u} = 1/2 = f_{uv,v}$ (so $f_{uv,u} + f_{uv,v} = 1$) and $d = \Delta/2$. The quantity $\sum_{uv \in E} f_{uv,v} = \deg(v)/2 \leq \Delta/2 = d$ enters every vertex $v \in V$ and thus $d^* \leq d \leq \Delta/2$. □

B Almost Unit Capacity Networks

Lemma 3.1 states that one should send as much flow as possible on the 'direct' way, the length-2 paths $s \to v \to t$ for $v \in V$ in flow networks. While this is intuitive, we provide a proof.

Proof (Lemma 3.1). For every $v \in V$, define $F_v := \min(c(s,v), c(v,t))$. Consider the feasible flow f^- in N where $f^-(s,v) = F_v = f^-(v,t)$ for all $v \in V$ and $f^-(u,v) = 0$ for $(u,v) \in E$. We will now reduce the capacities by these flow values to obtain the flow network N': Define $c' := c - f^-$.

The crucial idea is that any cut in N has exactly the capacity of the corresponding cut in N' plus the values F_v for $v \in V$.

Let M denote the value of the maximum flow in N. This is also the capacity of some minimum cut (S,T) in N. In N', the cut (S,T) has capacity

$$\begin{aligned} C'_{(S,T)} &= \sum_{v \in T} c'(s,v) + \sum_{v \in S} c'(v,t) + \sum_{\substack{u \in S \setminus \{s\} \\ v \in T \setminus \{t\}}} c'(u,v) \\ &= M - \sum_{v \in V} F_v. \end{aligned}$$

If (S,T) is also a minimum cut in N', its capacity must equal the maximum flow value of N'. We can then add the flow f^- to a maximum flow of N' to obtain a feasible flow for N with a value of $(M - \sum_{v \in V} F_v) + \sum_{v \in V} F_v = M$. It is thus a maximum flow.

It remains to show that (S,T) is indeed a minimum cut of N'. Assume otherwise, i.e. there exists a cut (S^*, T^*) with

$$C'_{(S^*,T^*)} < C'_{(S,T)} = C_{(S,T)} - \sum_{v \in V} F_v. \tag{B.1}$$

Thus,

$$C_{(S^*,T^*)} = C'_{(S^*,T^*)} + \sum_{v \in V} F_v \overset{\text{(B.1)}}{<} C_{(S,T)},$$

so (S,T) is not a minimum cut in N, which contradicts the assumption. □

By bounding the maximum flow in AUC-2 networks, we are able to prove that blocking flows can be computed in linear time. To compute a blocking flow, Dinitz's algorithm uses a depth-first search on the level network, which is constructed in a breadth-first search. The DFS performs 'advance' and 'backtrack' steps. Every time t is reached, as much flow as possible is pushed along the s-t path and saturated arcs on it are deleted.

Proof (Proposition 3.2). There are at most $2|V| + |E|$ arcs in the network, so the number D of deletions is at most $D \leq 2|V| + |E|$. The total number B of backtrack steps is at most the number of advance steps A, which is also an upper bound on the number of pushes P performed on the arcs. The number of advance steps on an arc is bounded by its capacity. Thus, we have

$$\begin{aligned} A &\leq \sum_{v \in V} c(s,v) + \sum_{(u,v) \in E} c(u,v) + \sum_{v \in V} c(v,t) \\ &= C_{s,t}(N) + \sum_{(u,v) \in E} c(u,v) \leq 2|E| + 2|E| = 4|E|. \end{aligned}$$

Therefore, a blocking flow can be found in $A + B + D + P \in \mathcal{O}(|E|)$ steps. □

The following analysis is analogous to that of Even and Tarjan [26]. Lemma B.1 applies to any network and remains unchanged.

LEMMA B.1. ([26]) *Let N be a flow network with maximum flow M, and let f be a flow. Then the maximum flow in the residual network $\tilde{N}$ is $M - |f|$.*

Proof. In the following, let S denote a set of nodes where $t \notin S \ni s$, and $(S, \bar{S})$ the set of arcs that go from S to $\bar{S}$. We have

$$\sum_{a \in (S,\bar{S})_{\tilde{N}}} \tilde{c}(a) = \sum_{a \in (S,\bar{S})_N} (c(a) - f(a)) + \sum_{a \in (\bar{S},S)_N} f(a).$$

Since

$$|f| = \sum_{a \in (S,\bar{S})_N} f(a) - \sum_{a \in (\bar{S},S)_N} f(a),$$

we have

$$\sum_{a \in (S,\bar{S})_{\tilde{N}}} \tilde{c}(a) = \sum_{a \in (S,\bar{S})_N} c(a) - |f|.$$

This implies that $(S, \bar{S})$ is a minimum cut of N if and only if $(S, \bar{S})$ is a minimum cut of $\tilde{N}$. The value of the minimum cut in N is M by the max-flow min-cut theorem, thus the value of the minimum cut in $\tilde{N}$ is $M - |f|$, which is the value of the maximum flow. □

Let us prove Lemma 3.2 and Theorem 3.1 in a detailed analysis.

LEMMA B.2. *Let N be an AUC-2 G-network with bounded maximum flow $0 \neq M \leq 2|E|$. For the zero flow, the distance from s to t is at most $\frac{6|E|}{M}$.*

Proof. Define

$$V_i := \{v \in V \cup \{s,t\} \mid v \text{ is at distance } i \text{ from } s\},$$

and let l denote the distance of t from s, where unreachable vertices ($i = \infty$) are not of interest. Let E_i denote the set of arcs from V_{i-1} to V_i. Every E_i defines a cut in the network. For $i = 2, ..., l-1$, we have

$$2|E_i| \geq M \tag{B.2}$$

because these arcs have a capacity of two at most and M is the value of the minimum cut. However, we may have $2|E_1|, 2|E_l| \ll M$ because the arcs may have a larger capacity than two. Therefore,

$$\begin{aligned} 2|E| &\geq \sum_{i=1}^{l} 2|E_i| \\ &\geq 2|E_1| + (l-2)M + 2|E_l| \geq (l-2)M, \end{aligned}$$

so we have $l \leq \frac{2|E|}{M} + 2 = \frac{2|E|+2M}{M} \leq \frac{6|E|}{M}$. □

THEOREM B.1. *Dinitz's algorithm runs in $\mathcal{O}(|E|^{3/2})$ time on an AUC G-network N with bounded total source and sink arc capacities $C_{s,t}(N) \leq 2|E|$.*

Proof. If $M \leq \sqrt{|E|}$, the result follows from Proposition 3.2 since every phase increases the flow by at least one. Otherwise, consider the phase in which the flow value F reaches the value $M - \sqrt{|E|}$. When this phase begins, we have $F < M - \sqrt{|E|}$. The residual network $\tilde{N}$ is an AUC-2 G-network with bounded maximum flow $\tilde{M} \leq 2|E|$. By Lemma B.1, its maximum flow is

$$\tilde{M} = M - F > M - (M - \sqrt{|E|}) = \sqrt{|E|}.$$

Since the flow in the residual network is initially zero, by Lemma B.2, the length of the shortest augmenting path satisfies

$$\tilde{l} \leq \frac{6|E|}{\tilde{M}} < \frac{6|E|}{\sqrt{|E|}} = 6\sqrt{|E|}.$$

The number of phases until this point is thus at most $6\sqrt{|E|}$, and since at most $\sqrt{|E|}$ phases are needed until completion, the total number of phases is less than $7\sqrt{|E|}$, thus the algorithm needs $\mathcal{O}(|E|^{3/2})$ steps by Proposition 3.2. □

LEMMA B.3. *Let N be an AUC-2 G-network with bounded maximum flow $0 \neq M \leq 2|E|$. For the zero flow, the distance from s to t is less than $\frac{(1+\sqrt{8})|V|}{\sqrt{M}}$.*

Proof. Define

$$V_i := \{v \in V \cup \{s,t\} \mid v \text{ is at distance } i \text{ from } s\}$$

and let l denote the distance of t from s, where unreachable vertices ($i = \infty$) are not of interest. We consider the cuts between V_i and V_{i+1} for $i = 0, ..., l$. Since M is the value of the minimum cut, the value $C(i, i+1)$ of the cut between V_i and V_{i+1} must be at least M. Therefore, we have $2(|V_i| \cdot |V_{i+1}|) \geq C(i, i+1) \geq M$ for $1 \leq i \leq l-2$ since every arc from V_i to V_{i+1} has a capacity of two at most. Therefore, for every $1 \leq i \leq l-2$, we have $|V_i| \geq \sqrt{M/2}$ or $|V_{i+1}| \geq \sqrt{M/2}$.

We now intend to sum over the cardinalities $|V_i|$ for $i = 0, ..., l$. We would like to argue that for any two successive sets, one set has at least $\sqrt{M/2}$ vertices. In the worst case, the summation sequence would alternate between values less than and greater or equal to $\sqrt{M/2}$. However, in our case it is possible that $|V_1|, |V_{l-1}| \ll \sqrt{M/2}$, since source and sink arc capacities can be larger than two. Thus,

$$\frac{l-1}{2} \cdot \sqrt{M/2} \leq 2 + \left\lfloor \frac{l-1}{2} \right\rfloor \cdot \sqrt{M/2} \leq \sum_{i=0}^{l} |V_i| \leq |V|.$$

Since $M \leq 2|E| \leq 2^{\frac{|V|(|V|-1)}{2}} < |V|^2$, we have $\sqrt{M} < |V|$. By applying this fact, we get

$$l \leq \frac{2|V|}{\sqrt{M/2}} + 1 = \frac{2\sqrt{2}|V| + \sqrt{M}}{\sqrt{M}} < \frac{\left(1+\sqrt{8}\right)|V|}{\sqrt{M}}.$$

□

THEOREM B.2. *Dinitz's algorithm runs in $\mathcal{O}(|V|^{2/3}|E|)$ time on an AUC-2 G-network with bounded total source and sink arc capacities $C_{s,t}(N) \leq 2|E|$.*

Proof. If $M \leq |V|^{2/3}$, the result follows since the flow increases at least by one per phase of Dinitz's algorithm. Otherwise, let F be the flow value of the phase in which the flow reaches the value $M - |V|^{2/3}$. We have $F < M - |V|^{2/3}$.

By Lemma B.1, the maximum flow in the residual network is

$$\tilde{M} = M - F > M - (M - |V|^{2/3}) = |V|^{2/3}.$$

Since the flow in the residual network is initially zero, we can apply Lemma B.3: The length of the shortest path satisfies

$$\tilde{l} < \frac{\left(1+\sqrt{8}\right)|V|}{\sqrt{\tilde{M}}} < \frac{\left(1+\sqrt{8}\right)|V|}{\sqrt{|V|^{2/3}}} = \left(1+\sqrt{8}\right)|V|^{2/3}.$$

Thus the number of phases up to this point is at most $(1+\sqrt{8})|V|^{2/3}$, and at most $|V|^{2/3}$ phases remain, for a total of $\mathcal{O}(|V|^{2/3})$ phases. □

C Faster Algorithms for Pseudoarboricity

Approximations can help speed up exact computation. We now give the final analysis omitted in the proof of Lemma 3.3.

Proof (Lemma 3.3, continued). We perform the binary search on an interval of integers, which needs $\mathcal{O}(\log L)$ tests for an interval length L.

$$L \leq (d-1) - \left\lceil \frac{d-1}{1+\delta} \right\rceil + 1 \leq (1+\delta)x - \frac{d-1}{1+\delta} + 1.$$

Now, we apply the bounds $x \leq d$ and $\delta > 0$ and obtain

$$\begin{aligned} L &\leq (1+\delta)x - \frac{x-1}{1+\delta} + 1 < \frac{(1+\delta)^2 x - x}{1+\delta} + 2 \\ &= \frac{(\delta^2+2\delta)x}{1+\delta} + 2 = \left(\delta + \frac{\delta}{1+\delta}\right) x + 2 \\ &< 2\delta x + 2. \end{aligned}$$

Thus, we need to perform $\mathcal{O}(\log(\delta x))$ tests in the binary search at most. □

We generalize Lemma 3.4 to 'relaxed' orientations. These are characterized by Cohen's LP (see Section 5).

LEMMA C.1. *Let (f_u, f_v, d) be a solution to Cohen's LP for some $d > d^*(G)$. Then for every vertex $v \in V$, the unit distance in this relaxed orientation to a vertex u with smaller relaxed indegree does not exceed* $\log_{d/d^*} |V|$.

Proof. Let v be an arbitrary vertex. Let k denote the minimum unit distance in $\vec{G}_{rel}$ from v to a vertex whose relaxed indegree is smaller than d. By distance in the relaxed orientation we mean the distance in the following BFS started in v: only edges may be traversed that are oriented to the vertex removed from the queue by more than zero. For example, an edge uv with $f_{uv,v} > 0$ may be traversed from v to u, an edge uv with $f_{uv,v} = 0$ may not be traversed from v to u.

Let $|V_i|$ denote the set of vertices which are at distance i at most from v. We show by induction that $|V_i| \geq (\frac{d}{d^*})^i$ for $i = 0, ..., k$. The claim holds for $i = 0$. Assume the induction hypothesis holds for some $i < k$. Let E_{i+1} denote the set of edges where both ends are in V_{i+1}.

Every vertex in V_i has a relaxed indegree of at least d (otherwise $i \geq k$, which contradicts the assumption) and since we have a relaxed d-orientation, its relaxed indegree is at most d, so it is exactly d. Thus

$$\begin{aligned} |E_{i+1}| &= \sum_{uw \in E_{i+1}} (f_{uw,u} + f_{uw,w}) \\ &\geq \sum_{u \in V_i} \sum_{\substack{uw \in E \\ f_{uw,u} > 0}} f_{uw,u} = \sum_{u \in V_i} \sum_{uw \in E} f_{uw,u} = d|V_i|. \end{aligned}$$

Since $\frac{|E_{i+1}|}{|V_{i+1}|} \leq d^*$, we obtain $|V_{i+1}| \geq \frac{d}{d^*}|V_i|$. By applying the induction hypothesis, the claim is shown for all $i = 0, ..., k$.

Because $|V_k| \leq |V|$, we have $(\frac{d}{d^*})^k \leq |V|$, which concludes the proof. □

The lemma allows us to directly approximate d^* with shortest-augmenting path algorithms on the re-orientation network with possibly non-integral guesses. Note however that runtime analyses of flow algorithms often require integral capacities.

D Linear Programs

As mentioned in Section 5, every solution to Cohen's LP (whose polyhedron we call $\mathcal{P}_C$) is a maximum flow in Goldberg's flow network with parameter d.

PROPOSITION D.1. *Every solution $(f_u, f_v, d) \in \mathcal{P}_C$ is a maximum flow in Goldberg's network for guess $g = d$.*

Proof. Let $(\{s\}, V \cup \{t\})$ be a minimum cut of Goldberg's network for some guess $g \geq 0$. Then, we have $g \geq d^*$ [3, Theorem 1]. Therefore, there is a Cohen solution $(f_u, f_v, g) \in \mathcal{P}_C$. Let $in(v)$ denote the ingoing 'orientation flow' in Cohen's network. We set the flow in Goldberg's network to be $f(u,v) = f_{uv,v}$, $f(s,v) = |E|$ and $f(v,t) = |E| + 2in(v) - \deg(v)$ for all $u, v \in V$. We first show that the flow is feasible. The sink arc capacity constraints are fulfilled since we have $in(v) \leq g$,

$$\begin{aligned} f(v,t) &= |E| + 2in(v) - \deg(v) \\ &\leq |E| + 2g - \deg(v) = c(v,t). \end{aligned}$$

All other capacity constraints are trivially satisfied. Every vertex v receives $in(v)$ from its neighbors and sends $\deg(v) - in(v)$ to its neighbors. Furthermore, it receives $|E|$ from the source and sends $|E| + 2in(v) - \deg(v)$ to the sink. Flow conservation is fulfilled since $|E| + in(v) = (\deg(v) - in(v)) + (|E| + 2in(v) - \deg(v))$. Thus the flow f is feasible. It is also a maximum flow of value $|V| \cdot |E|$ since all source arcs are saturated. □