Approximation Scheme for Lowest Outdegree Orientation and Graph Density Measures

Łukasz Kowalik[⋆1,2]

[1] Institute of Informatics, Warsaw University, Warsaw, Poland
[2] Max-Planck-Institute für Informatik, Saarbrücken, Germany
kowalik@mimuw.edu.pl

Abstract. We deal with the problem of finding such an orientation of a given graph that the largest number of edges leaving a vertex (called the outdegree of the orientation) is small.
For any $\varepsilon \in (0, 1)$ we show an $\tilde{O}(|E(G)|/\varepsilon)$ time algorithm[3] which finds an orientation of an input graph G with outdegree at most $\lceil (1 + \varepsilon)d^* \rceil$, where d^* is the maximum density of a subgraph of G. It is known that the optimal value of orientation outdegree is $\lceil d^* \rceil$.
Our algorithm has applications in constructing labeling schemes, introduced by Kannan *et al.* in [18] and in approximating such graph density measures as arboricity, pseudoarboricity and maximum density. Our results improve over the previous, 2-approximation algorithms by Aichholzer *et al.* [1] (for orientation / pseudoarboricity), by Arikati *et al.* [3] (for arboricity) and by Charikar [5] (for maximum density).

1 Introduction

In this paper we deal with approximating lowest outdegree orientation, pseudoarboricity, arboricity and maximum density. Let us define these notions as they are not so widely used.

Let $G = (V, E)$ be a graph. An *orientation* of G is a digraph $\overrightarrow{G} = (V, \overrightarrow{E})$ that is obtained from G by replacing every undirected edge uv by an arc, i.e., (u, v) or (v, u). The *outdegree* of an orientation is the largest of its vertices' outdegrees. In this paper we focus on the problem of finding for a given graph its orientation with minimum outdegree. We will call it a *lowest outdegree orientation*. This problem is closely related to computing pseudoarboricity and maximum density of graphs.

Density of a graph $G = (V, E)$, denoted by $d(G)$, is defined as $d(G) = |E|/|V|$, i.e., it is half of its average degree. In the *Densest Subgraph Problem*, given graph G one has to find its subgraph G^* such that any nonempty subgraph H of G satisfies $d(H) \leq d(G^*)$. The number $d(G^*)$ will be called *maximum density* of graph G, and we will denote it by $d^*(G)$. As it was shown by Charikar [5], the linear program for Densest Subgraph Problem is dual to the relaxation of the integer program for finding the lowest outdegree orientation. Moreover, it follows from a theorem by Frank and

⋆ Supported in part by KBN grant 4T11C04425.
[3] The $\tilde{O}(\cdot)$ notation ignores logarithmic factors.

Gyárfás [11] that $\lceil d^*(G) \rceil$ equals the outdegree of the lowest outdegree orientation of G.

A *pseudotree* is a connected graph containing at most one cycle. *Pseudoforest* is a union of vertex disjoint pseudotrees. *Pseudoarboricity* of graph G, denoted as $P(G)$, is the smallest number of pseudoforests needed to cover all edges of G. As it was noticed by Picard and Queyranne [22], $P(G) = \lceil d^*(G) \rceil$, which combined with the theorem of Frank and Gyárfás implies that pseudoarboricity equals the outdegree of the lowest outdegree orientation. This can be also easily proved directly (see Section 2).

Arboricity of graph G, denoted as $\mathrm{arb}(G)$, is the smallest number of forests needed to cover all edges of G. A classic theorem by Nash-Williams [21] says that arboricity is equal to $\max_J \lceil |E(J)|/(|V(J)|-1) \rceil$ where J is any subgraph of G with $|V(J)| \geq 2$ vertices and $|E(J)|$ edges. Using this fact it is easy to show (see [22]) that $P(G) \leq \mathrm{arb}(G) \leq P(G)+1$.

Applications Arboricity is the most often used measure of graph sparsity. Complexity of many graph algorithms depends heavily on the arboricity of the input graph – see e.g. [6, 4, 8].

Kannan et al. [18] noticed that for any n-vertex graph of arboricity k one can label its vertices using at most $(k+1)\log n$ bits for each label in such a way that adjacency of any pair of vertices can be verified using merely their labels. They call it a $(k+1)$-*labeling scheme*. It is achieved as follows: (1) number the vertices from 1 to n, (2) find a partition of the graph into k forests, (3) in each tree in each forest choose a root, and (4) assign each vertex a label containing its number and the numbers of its parents in the at most k trees it belongs to. Then to test adjacency of vertices u and v it suffices to check whether u is the parent of v in some tree or vice versa.

Chrobak and Eppstein [7] observed that in order to get the labeling schemes one can use orientations instead the partition into forests. Then each vertex v stores in its label the numbers of endpoints of the arcs leaving v. As for any graph G, $P(G) \leq \mathrm{arb}(G)$ this is a little bit more efficient approach. Then the problem of building a $(P(G)+1)$-labeling scheme reduces to the problem of finding a lowest degree orientation.

It should be noted that low outdegree orientations are used as a handy tool in many algorithms, see e.g., [2, 19, 9].

Related Work Throughout the whole paper m and n denote the number of edges and vertices of the input graph, respectively.

The problem of computing pseudoarboricity and the related decomposition was first raised by Picard and Queyranne [22] who applied network flows and obtained $\mathcal{O}(nm\log^3 n)$ algorithm by using the maximum flow algorithm by Galil and Naamad [14]. It was improved by Gallo, Grigoriadis and Tarjan [15] to $\mathcal{O}(nm\log(n^2/m))$ by using parametric maximum flow. Next Gabow and Westermann [13] applied their matroid partitioning algorithm for the k-pseudoforest problem – the problem of finding in a given graph k edge-disjoint forests containing as many edges as possible. Their algorithm works in $\mathcal{O}(\min\{(kn')^{3/2}, k(n')^{5/3}\})$ time where $n' = \min\{n, m/k\}$. As pseudoarboricity is at least m/n it gives an $\mathcal{O}(m\min\{m^{1/2}, n^{2/3}\})$-time algorithm which verifies whether a given graph has pseudoarboricity at most k and if the answer is yes computes the relevant pseudoforest partition. Using binary search pseudoarboricity p can be computed in $\mathcal{O}(m\min\{m^{1/2}, n^{2/3}\}\log p)$ time. However, if the pseudoforest partition is not needed, they show also that this value can be found in $\mathcal{O}(m \min\{(m \log n)^{1/2}, (n \log n)^{2/3}\})$ time. For the related problem of finding arboricity and the relevant forest partition they describe an $\mathcal{O}(m^{3/2}\sqrt{\log n})$ algorithm.

Finally, Aichholzer et al. [1] claimed (without giving details) that one can solve the equivalent lowest outdegree orientation problem in $\mathcal{O}(m^{3/2} \log p)$ time by using Dinic's algorithm.

Since the problem seems to be closely related with network flows and matroid partitions it can be hard to get a near-linear algorithm for it. Hence one can consider approximation algorithms. Arikati et al. [3] showed a simple linear-time 2-approximation algorithm for computing arboricity and the corresponding partition into forests. Independently, Aichholzer et al. [1] showed a 2-approximation algorithm for the problem of finding lowest outdegree orientation (and hence also pseudoarboricity). In fact, these two algorithms are the same. Both can be viewed as finding for a given graph G its acyclic orientation of outdegree at most $2P(G)$.

Recently, Gabow [12] considered a related problem of orienting as many edges as possible subject to upper bounds on the indegree and outdegree of each vertex. He proves that the problem is MAXSNP-hard and shows a $3/4$-approximation algorithm.

For the densest subgraph problem the state of art is very similar to computing pseudoarboricity. A paper of Goldberg [16] contains a reduction to a network flow problem, which combined with the algorithm by Goldberg and Rao [17] gives an algorithm with time complexity $\tilde{\mathcal{O}}(m \min\{n^{2/3}, m^{1/2}\})$. On the other hand, Charikar [5] showed a simple linear-time 2-approximation algorithm.

Our Results We show an algorithm which, given a number $\varepsilon > 0$ and a graph with maximum density d^*, finds a d-orientation so that $d \leq \lceil (1+\varepsilon)d^* \rceil$. In other words, it is an approximation scheme with additional additive error (caused by rounding up) bounded by 1. For $0 < \varepsilon < 1$ the algorithm works in $\mathcal{O}(m \log n \max\{\log d^*, 1\}\varepsilon^{-1})$ time.

As $P(G) \leq \mathrm{arb}(G) \leq P(G) + 1$ and $P(G) = \lceil d^*(G) \rceil$ it is not surprising that our algorithm can be also used for efficient approximating arboricity and maximum density – for both these problems we get an approximation scheme with an additional small additive error (2 for arboricity, 1 for maximum density). In Section 2 we also note that finding a partition of edges of a graph into d pseudoforests is equivalent to the problem of finding an orientation with outdegree d. Thus our algorithms apply also to the pseudoforest partition problem.

In the particular case of *sparse graphs*, i.e., graphs of bounded arboricity, the running time of our algorithm is $\mathcal{O}((n \log n)/\varepsilon)$, as then $d^* = \mathcal{O}(1)$ and $m = \mathcal{O}(n)$. It is worth noting that for sparse graphs our algorithm can be used to efficiently find an orientation with outdegree $\lceil d^* + \delta \rceil$, for $\delta > 0$. Alternatively, we can use it for approximating arboricity (additive error $1 + \lceil \delta \rceil$) and maximum density (additive error $1 + \delta$). This can be done in $\mathcal{O}(m \log n \max\{\log d^*, 1\} \max\{\frac{d^*}{\delta}, 1\})$ time, which is $\mathcal{O}(n \log n \max\{\delta^{-1}, 1\})$ for sparse graphs. In particular, for sparse graphs this gives $\mathcal{O}(n \log n)$ time approximation algorithms with additive error 1 (for lowest outdegree orientation / pseudoarboricity), or 2 (for arboricity and maximum density).

The idea of our approximation algorithms is very simple. We start Dinic's maximum flow algorithm in some network which depends on the input graph and some parameter

d. We stop it when augmenting paths grow too long. If d is greater, but not very close to the maximum density d^* we show that the augmenting paths will never grow too long and then we obtain a d-orientation. Otherwise we know d is too close to d^* — closer than we need. In order to find the smallest value of d such that the augmenting paths are always short we use binary search.

2 Preliminaries

We say that $\overrightarrow{G}$ is a d-orientation when vertex outdegrees in $\overrightarrow{G}$ do not exceed d. We assume that the reader is familiar with basic concepts concerning network flow algorithms. For details see e.g. [20]. Let us recall here only some basic notions.

Let $G = (V, E)$ be a directed graph with two special vertices s (called source) and t (called sink). Each arc of G is assigned a number called *capacity*. More precisely, *capacity* is a function $c : V^2 \to \mathbb{R}_{\geq 0}$ such that for $(v, w) \notin E$, $c(v, w) = 0$. Graph G with the capacity function c is called a *network*. *Flow* in a network G is any function $f : V^2 \to \mathbb{R}$ such that for any $u, v \in V$ (1) $f(u, v) \leq c(u, v)$, (2) $f(u, v) = -f(v, u)$, (3) if $v \neq s, t$, $\sum_{x\in V} f(v, x) = 0$. The *value* of flow f, denoted by $|f|$, is the value of $\sum_{x\in V} f(s, x)$. A *maximum flow* is a flow with largest possible value. For network G and flow f the *residual capacity* is a function $c_f : V^2 \to \mathbb{R}_{\geq 0}$ such that $c_f(u, v) = c(u, v) - f(u, v)$. The graph with vertex set V containing edge (u, v) if and only if $c_f(u, v) > 0$ is denoted as G_f. Graph G_f with c_f as capacity function is called a *residual network*. *An augmenting path* is any path from s to t in the residual network. Edge (u, v) of graph G is called *augmented* when $f(u, v) = c(u, v)$.

Below we show an important relation between partitions into pseudoforests and orientations.

Proposition 1. *The problems of finding p-orientation and partition into p pseudoforests are equivalent, i.e., from a given p-orientation of some graph one can find a partition of edges of this graph into p pseudoforests and vice versa. Both conversions take time linear in the number of edges.*

Proof. Every pseudotree has a 1-orientation, as it suffices to remove any edge of the cycle (if there is one), choose one of its ends as the root, orient all edges toward the root and finally add the removed edge oriented from the root to the other endpoint. Thus given a decomposition of a graph into p pseudoforests we can find its p-orientation in linear time.

Conversely, consider a connected graph G with a 1-orientation. We will show that G is a pseudotree. G has at least $|V(G)|-1$ edges since it is connected, and at most $|V(G)|$ edges since it has 1-orientation. If it has $|V(G)| - 1$ then it is a tree. If it has $|V(G)$ edges it contains a cycle. After removing any edge of this cycle we get a connected graph G' with $|V(G')| - 1$ edges. Hence G' is a tree which implies that G has precisely one cycle. It follows that a graph with a 1-orientation is a pseudoforest.

Now, given a p-orientation, for each vertex we remove any of the edges leaving it. Then we obtain a $(p-1)$-orientation and the removed edges form a 1-orientation, which is a pseudoforest when we forget about edge orientations. After repeating this step p times we obtain the desired decomposition into p pseudoforests. The whole process also takes linear time. □

The above proposition implies that finding pseudoarboricity and the corresponding partition of edges into pseudoforests is equivalent to finding the lowest degree orientation.

3 Reduction to a Flow Problem

Here we present a reduction of finding a d-orientation of a given graph (if it exists) to finding a maximum flow in some network. Other reductions are used in [22, 1].

Let $G = (V, E)$ be a graph and let d be a positive integer. Let $\overrightarrow{G}$ be an arbitrary orientation of G (we will call it *the initial orientation*). We build a network $\tilde{G}^d = (\tilde{V}, \tilde{E})$ with capacity function c^d as follows. Set $\tilde{V}$ contains all vertices from V and two new vertices s (source) and t (sink). Set $\tilde{E}$ contains all edges from $E(\overrightarrow{G})$, each with capacity 1, an edge (s, v) with capacity $\mathrm{outdeg}(v) - d$ for each vertex v with outdegree in $\overrightarrow{G}$ greater than d and an edge (v, t) with capacity $d - \mathrm{outdeg}(v)$ for each vertex v with outdegree in $\overrightarrow{G}$ smaller than d. Let $\tilde{G}^d_f$ denote the residual network for flow f. Note the following proposition.

Proposition 2. *For any integral flow f in network $\tilde{G}^d$, the subgraph of the residual network $\tilde{G}^d_f$ induced by set V is an orientation of graph G.* □

Let us denote the subgraph described above by $\overrightarrow{G}_f$. From now on we assume that flows in $\tilde{G}^d$ are integral.

Lemma 1. *Let f be any flow in network $\tilde{G}^d$. There is an edge (s, v) in $\tilde{G}^d_f$ if and only if $\mathrm{outdeg}_{\overrightarrow{G}_f}(v) > d$. Also, there is an edge (v, t) in $\tilde{G}^d_f$ if and only if $\mathrm{outdeg}_{\overrightarrow{G}_f}(v) < d$.*

Proof. Let $\overrightarrow{G}$ be the initial orientation of G and let v be an arbitrary vertex in V. Clearly, $\mathrm{outdeg}_{\overrightarrow{G}_f}(v) = \mathrm{outdeg}_{\overrightarrow{G}}(v) + \sum_{w \in V} f(w, v)$. As f is a flow, $0 = \sum_{w \in \tilde{V}} f(w, v) = \sum_{w \in V} f(w, v) + f(s, v) + f(t, v)$. Hence,

$$\mathrm{outdeg}_{\overrightarrow{G}_f}(v) = \mathrm{outdeg}_{\overrightarrow{G}}(v) - f(s, v) + f(v, t). \tag{1}$$

Now, if $c(s, v) = c(v, t) = 0$ then $f(s, v) = f(v, t) = 0$ and we see that both (s, v) and (v, t) are not in $\tilde{G}^d_f$ and $\mathrm{outdeg}_{\overrightarrow{G}_f}(v) = d$.

If $c(s, v) > 0$ then $c(v, t) = 0$ and further $f(v, t) = 0$. Hence, by (1), we have $c(s, v) - f(s, v) = \mathrm{outdeg}_{\overrightarrow{G}}(v) - d - f(s, v) = \mathrm{outdeg}_{\overrightarrow{G}_f}(v) - d$. Then $c(s, v) - f(s, v) > 0$ if and only if $\mathrm{outdeg}_{\overrightarrow{G}_f}(v) - d > 0$, which is equivalent to the first part of the lemma. Also, since $c(v, t) = 0$ and $c(t, v) = 0$ we get $(v, t) \notin E(\tilde{G}^d_f)$. Moreover, $0 \leq c(s, v) - f(s, v) = \mathrm{outdeg}_{\overrightarrow{G}_f}(v) - d$ which implies that the second part of the lemma also holds in this case. The case $c(s, v) < 0$ can be verified analogously. □

Corollary 1. *There is an augmenting path $s v_1 v_2 \ldots v_k t$ in the residual network $\tilde{G}^d_f$ iff there is a path $v_1 v_2 \ldots v_k$ in $\overrightarrow{G}_f$ such that $\mathrm{outdeg}_{\overrightarrow{G}_f}(v_1) > d$, $\mathrm{outdeg}_{\overrightarrow{G}_f}(v_k) < d$.*

Let $c^d(s, V - s)$ denote the capacity of the $(\{s\}, V \setminus \{s\})$ cut, i.e., $c^d(s, V - s) = \sum_{v \in V} c^d(s, v)$.

Theorem 1. *Let G be a graph and let f be a maximum flow in network $\tilde{G}^d$. There exists a d-orientation of G if and only if $|f| = c^d(s, V - s)$. Moreover, when $|f| = c^d(s, V - s)$ then $\overrightarrow{G}_f$ is a d-orientation of G.*

Proof. Assume that there is a d-orientation of G and the maximum flow f in $\tilde{G}^d$ is smaller than $c^d(s, V - s)$. Then at least one edge leaving s, say (s, v) is not augmented. Then from Lemma 1 $\mathrm{outdeg}_{\overrightarrow{G}_f}(v) > d$. Let $W \subseteq V$ denote the set of vertices reachable from v in $\overrightarrow{G}_f$. Since there is a d-orientation of G, graph $G[W]$ contains at most $d|W|$ edges. If W contained no vertex of outdegree smaller than d then graph $G[W]$ would contain more than $d|W|$ edges, which would be a contradiction. Hence W contains a vertex w such that $\mathrm{outdeg}_{\overrightarrow{G}_f}(w) < d$ and by Corollary 1 there is an augmenting path which contradicts the maximality of flow f.

Conversely, if f is a flow in $\tilde{G}^d$ of value $c^d(s, V - s)$ there are no edges leaving s in the residual network and Lemma 1 implies that outdegrees in $\overrightarrow{G}_f$ do not exceed d. □

In order to analyze our approximation algorithm we will use the following lemma. The lemma and its proof is analogous to Lemma 2 in [4] (however, our Lemma 2 implies Lemma 2 in [4] and not vice-versa, hence we include the proof for completeness).

Lemma 2. *Let $\overrightarrow{G}$ be a d-orientation of some n-vertex graph G of maximum density d^* and let $d > d^*$. Then for any vertex v the distance in $\overrightarrow{G}$ to a vertex with outdegree smaller than d does not exceed $\log_{d/d^*} n$.*

Proof. Let v be an arbitrary vertex and let k be the distance from v to a vertex with outdegree smaller than d. For every $i = 0, \ldots, k$ let V_i be the set of vertices at distance at most i from v. We will show by induction that for each $i = 0, \ldots, k$, $|V_i| \geq (\frac{d}{d^*})^i$. We see that this inequality holds for $i = 0$. For the induction step assume that $i < k$. Let E_{i+1} be the set of edges with both ends in V_{i+1}. We see that exactly $d|V_i|$ edges leave V_i. Since all these edges belong to E_{i+1} it gives us $|E_{i+1}| \geq d|V_i|$. As $\frac{|E_{i+1}|}{|V_{i+1}|} \leq d^*$ we get $|V_{i+1}| \geq \frac{d}{d^*}|V_i|$. After applying the induction hypothesis we get the desired inequality. Then since $|V_k| \leq n$ we get $(\frac{d}{d^*})^k \leq n$ which ends the proof. □

As an immediate consequence of Corollary 1 and Lemma 2 we get the following corollary.

Corollary 2. *Let G be an n-vertex graph of maximum density d^* and let for some integer $d > d^*$, $\tilde{G}^d$ be the corresponding network with some flow f. If $\tilde{G}^d$ contains an augmenting path then it contains an augmenting path of length at most $2 + \log_{d/d^*} n$.*

4 Approximation Algorithm

Let us now briefly recall Dinic's algorithm. Details can be found in many textbooks, e.g. [20]. Dinic's algorithm begins with the empty flow f. It consists of a sequence of *phases*. In the beginning of each phase it builds a *layered network*, i.e., a subgraph of the residual network containing only edges of shortest paths from source s to sink t. The goal of each phase is to find a *blocking flow* in the layered network, i.e., such flow that each s,t-path in the layered network contains an augmented edge. In the end of each phase the blocking flow is added to flow f.

Dinic's algorithm finds the blocking flow by finding a number of augmenting paths, each time sending maximal amount of flow through the path. To find such path it uses the following method. Start from the empty path. Let v be the end of the path p found so far. If $v = t$ an augmenting path is found. If there is an edge leaving v add it to path p (this step is called *advance*). Otherwise remove the last edge of path p from both the layered network and p (this step is called *retreat*).

It is known that Dinic's algorithm finds a blocking flow in unit capacity networks in linear time (see e.g. [10]). It is not surprising that it is similarly fast in network $\tilde{G}^d$, which is "almost unit capacity". For completeness, below we give a proof.

Proposition 3. *For any graph G with m edges the Dinic's algorithm finds a blocking flow in network $\tilde{G}^d$ in $\mathcal{O}(m)$ time.*

Proof. The number of advance steps after which the sink t is reached is equal to the number of augmenting paths found, which is bounded by the value of maximum flow, which in turn is bounded by m. The total number of other advance steps is bounded by the sum of relevant edge capacities, i.e., $\sum_v c^d(s,v) + \sum_{v,w\in V} c^d(v,w) \leq \sum_v \mathrm{outdeg}(v) + \sum_{v,w\in V} c^d(v,w) = 2m$. The number of retreat steps is bounded by the number of edges. We see that the total number of advance and retreat steps is at most $4m$. □

Let us also recall another crucial property of Dinic's algorithm (see e.g. [20]):

Proposition 4. *After each phase of Dinic's algorithm the length of the shortest augmenting path increases.*

Now let us describe our main result, algorithm ORIENT(ε) which finds orientation of a given graph with outdegree close to optimal.

Algorithm 4.1 TEST(k,d)

1: Build $\tilde{G}^d$
2: **while** $\mathrm{dist}_{\tilde{G}^d_f}(s,t) \leq k$ **do**
3: Run another phase of Dinic's algorithm
4: **if** $|f| = c^d(s, V-s)$ **then return** $\overrightarrow{G}_f$ **else return** "FAIL"

We will use a subroutine TEST(k,d). It builds network $\tilde{G}^d$, and runs the Dinic's algorithm until it finishes (i.e. when there is no augmenting path) or the augmenting paths become longer than k. If the resulting flow has value $c^d(s, V-s)$, it returns an orientation $\overrightarrow{G}_f$. Otherwise it returns "FAIL" message. As an immediate consequence of propositions 3 and 4 we get the following proposition.

Proposition 5. *Algorithm* TEST(k,d) *works in* $\mathcal{O}(km)$ *time.* □

Lemma 3. *Let* G *be a graph with maximum density* d^* *and let* $d \geq \lceil (1+\varepsilon)d^* \rceil$ *for some* $\varepsilon > 0$. *Then* TEST$(2 + \log_{1+\varepsilon} n,\ d)$ *returns a* d*-orientation of* G.

Proof. As $\varepsilon > 0$ it follows that $d > d^*$ and by Corollary 2 if there is an augmenting path, there is an augmenting path of length at most $2 + \log_{d/d^*} n$, which is not greater than $2 + \log_{1+\varepsilon} n$. Hence the **while** loop is stopped when there is no augmenting path, i.e., $\mathrm{dist}_{\tilde{G}^d_f}(s,t) = \infty$, which implies that a maximum flow f is found. As $d \geq \lceil (1+\varepsilon)d^* \rceil \geq \lceil d^* \rceil = P(G)$, there exists a d-orientation of G, so by Theorem 1, $|f| = c^d(s, V - s)$ and $\overrightarrow{G}_f$ is a d-orientation. It establishes the proof. □

Algorithm 4.2 ORIENT(ε)

$d_1 \leftarrow 0$; $d_2 \leftarrow 1$
while TEST $(2 + \log_{1+\varepsilon} n, d_2)$ = "FAIL" **do**
  $d_1 \leftarrow d_2$; $d_2 \leftarrow 2d_2$
while $d_1 < d_2$ **do**
  $d' = \lceil \frac{d_1+d_2}{2} \rceil$
  if TEST $(2 + \log_{1+\varepsilon} n, d')$ = "FAIL" **then** $d_1 \leftarrow d'$ **else** $d_2 \leftarrow d'$
return the orientation returned by the last call of TEST

Algorithm ORIENT(ε) uses binary search to find an integer d such that TEST$(2 + \log_{1+\varepsilon} n,\ d-1)$ returns "FAIL" message, while TEST$(2 + \log_{1+\varepsilon} n,\ d)$ does not. (Note that it may happen that $d < \lceil (1+\varepsilon)d^* \rceil$). It returns the d-orientation returned by the relevant call of TEST. Now we state the main result of the paper.

Theorem 2. *Let* G *be any graph of maximum density* d^*. *For any* $\varepsilon > 0$ *algorithm* ORIENT(ε) *finds a* d*-orientation of* G *such that* $d \leq \lceil (1+\varepsilon)d^* \rceil$. *Its time complexity is* $\mathcal{O}(m \log n \max\{\log d^*, 1\} \max\{\varepsilon^{-1}, 1\})$.

Proof. Correctness of the algorithm is an immediate consequence of Lemma 3. By Proposition 5 each call of TEST$(2 + \log_{1+\varepsilon} n, d)$ subroutine takes $\mathcal{O}(m \log_{1+\varepsilon} n) = \mathcal{O}(m(\log n)(\log(1+\varepsilon))^{-1})$ time. By Taylor expansion, for $\varepsilon < 1$, $\ln(1+\varepsilon) = \varepsilon + \mathcal{O}(\varepsilon^2)$. Hence each call of TEST routine in algorithm ORIENT takes time bounded by $\mathcal{O}(m(\log n)\max\{\varepsilon^{-1}, 1\})$. Theorem 2 implies that ORIENT(ε) makes at most $\mathcal{O}(\lceil \log \lceil (1+\varepsilon)d^* \rceil \rceil) = \mathcal{O}(\max\{\log d^*, 1\})$ calls of subroutine TEST. Hence we get the claimed time bound. □

4.1 Approximating Graph Density Measures

Using our algorithm one can approximate efficiently graph density measures. The details are given in the following theorem.

Theorem 3. *Let* G *be any graph of maximum density* d^*. *For any* $\varepsilon > 0$ *there are algorithms with time complexity* $\mathcal{O}(m \log n \max\{\log d^*, 1\} \max\{\varepsilon^{-1}, 1\})$ *for the following problems:*

(i) (pseudoarboricity approximation) Finding a partition of G into $\tilde{d}$ pseudoforests so that $\tilde{d} \leq \lceil(1+\varepsilon)d^*\rceil \leq (1+\varepsilon)P(G)+1$.
(ii) (arboricity approximation) finding a number $\tilde{a}$ such that there exists a partition of G into $\tilde{a}$ forests so that $\tilde{a} \leq (1+\varepsilon)\mathrm{arb}(G)+2$.
(iii) (densest subgraph approximation) finding a number $\tilde{d^*}$ such that G contains a subgraph of density at least $\tilde{d^*}$ so that $\tilde{d^*} \geq (1-\varepsilon)d^* - 1$.

Proof. Part (i) follows immediately from Proposition 1 and Theorem 2.

To construct the algorithm described in (ii) it suffices to find the number $\tilde{d}$ using part (i) and report $\tilde{a} = \tilde{d}+1$. Since $P(G) \leq \mathrm{arb}(G)$, the claimed bound follows. The relevant partition into forests exists because $\tilde{a} \geq P(G)+1 \geq \mathrm{arb}(G)$.

Similarly, for part (iii) we also apply algorithm from part (i), but using different value of ε, namely using $\varepsilon' = \varepsilon/(1-\varepsilon)$. Since $\max\{(\varepsilon')^{-1},1\} = \max\{\varepsilon^{-1}-1,1\} \leq \max\{\varepsilon^{-1},1\}$, the algorithm works in the claimed time. Then we report $\tilde{d^*} = (\tilde{d}-1)/(1+\varepsilon')$. Since $\tilde{d} \leq \lceil(1+\varepsilon')d^*\rceil$, we see that $\tilde{d^*} \leq d^*$, hence there is a subgraph of density at least $\tilde{d^*}$. Finally, because $\tilde{d} \geq P(G) \geq d^*$, we get $\tilde{d^*} \geq \frac{d^*}{1+\varepsilon'} - \frac{1}{1+\varepsilon'} = (1-\varepsilon)d^* - (1-\varepsilon) > (1-\varepsilon)d^* - 1$. $\square$

4.2 Approximation with Additive Error

Now we observe that for sparse graphs our algorithm can be used to efficiently find an orientation with outdegree $\lceil d^*+\delta\rceil$, for $\delta > 0$. To this end one finds a d'-orientation, $d^* < d' < \frac{3}{2}d^*$ using algorithm ORIENT($\frac{3}{2}$). If $\delta \geq d'$ the algorithm stops and returns the d'-orientation found. Otherwise it calls algorithm ORIENT($\frac{\delta}{d'}$). Clearly the second call returns an orientation with outdegree $\lceil(1+\frac{\delta}{d'})d^*\rceil \leq \lceil(1+\frac{\delta}{d^*})d^*\rceil = \lceil d^*+\delta\rceil$. Time complexity is $\mathcal{O}(m\log n \max\{\log d^*,1\}\max\{\frac{d^*}{\delta},1\})$, which for sparse graphs can be rewritten as $\mathcal{O}(n\log n\max\{\delta^{-1},1\})$. Similarly as in Theorem 3 we obtain also algorithms with the same time complexity for approximating pseudoarboricity (additive error $\lceil\delta\rceil$), arboricity (additive error $1+\lceil\delta\rceil$), and maximum density (additive error $1+\delta$).

5 Further Research

We showed how to efficiently approximate *the values* of arboricity and maximum density. It is very natural to ask for near-linear algorithms for finding the relevant decomposition into forests and the relevant dense subgraph. In the context of the first problem it is particularly interesting whether there is a fast algorithm which transforms a decomposition of a graph into d pseudoforests to a decomposition into $d+1$ forests (or, if this is infeasible, then into $\alpha \cdot d$ forests, for some $\alpha < 2$).

References

1. O. Aichholzer, F. Aurenhammer, and G. Rote. Optimal graph orientation with storage applications. SFB-Report F003-51, SFB 'Optimierung und Kontrolle', TU Graz, Austria, 1995.
2. N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

3. S. R. Arikati, A. Maheshwari, and C. D. Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discrete Appl. Math.*, 78(1-3):1–16, 1997.
4. G. S. Brodal and R. Fagerberg. Dynamic representations of sparse graphs. In *Proc. 6th Int. Workshop on Algorithms and Data Structures (WADS'99)*, volume 1663 of *LNCS*, pages 342–351, 1999.
5. M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proc. 13th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'00)*, volume 1913 of *LNCS*, pages 84–95, 2000.
6. N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.
7. M. Chrobak and D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243–266, 1991.
8. D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, 1994.
9. D. Eppstein. All maximal independent sets and dynamic dominance for sparse graphs. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 451–459, 2005.
10. S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975.
11. A. Frank and A. Gyárfás. How to orient the edges of a graph? In *Combinatorics Volume I (Proc. of the Fifth Hungarian Colloquium on Combinatorics, Keszthely, 1976, A. Hajnal, V. T. Sós, eds.)*, pages 353–364, Amsterdam, 1976. North-Holland.
12. H. Gabow. Upper degree-constrained partial orientations. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, 2006.
13. H. Gabow and H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. In *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 407–421, New York, NY, USA, 1988. ACM Press.
14. Z. Galil and A. Naamad. An $O(EV \log^2 V)$ algorithm for the maximal flow problem. *J. Comput. System Sci.*, 21:203–217, 1980.
15. G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.
16. A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.
17. A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, page 2, Washington, DC, USA, 1997. IEEE Computer Society.
18. S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 334–343, New York, NY, USA, 1988. ACM Press.
19. Ł. Kowalik and M. Kurowski. Shortest path queries in planar graphs in constant time. In *Proc. 35th Symposium on Theory of Computing (STOC'03)*, pages 143–148. ACM, June 2003.
20. D. C. Kozen. *The design and analysis of algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
21. C. S. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39:12, 1964.
22. J.-C. Picard and M. Queyranne. A network flow solution to some nonlinear 0-1 programming problems with application to graph theory. *Networks*, 12:141–159, 1982.