Algorithmica 2018
doi: 10.1007/s00453-018-0528-0

A Simple Greedy Algorithm for Dynamic Graph Orientation

Edvin Berglin · Gerth Stølting Brodal

Abstract *Graph orientations with low out-degree* are one of several ways to efficiently store sparse graphs. If the graphs allow for insertion and deletion of edges, one may have to *flip* the orientation of some edges to prevent blowing up the maximum out-degree. We use arboricity as our sparsity measure. With an immensely simple greedy algorithm, we get parametrized trade-off bounds between out-degree and worst case number of flips, which previously only existed for amortized number of flips. We match the previous best worst-case algorithm (in $\mathcal{O}(\log n)$ flips) for almost all values of arboricity and beat it for either constant or super-logarithmic arboricity. We also match a previous best amortized result for at least logarithmic arboricity, and give the first results with worst-case $\mathcal{O}(1)$ and $\mathcal{O}\big(\sqrt{\log n}\big)$ flips nearly matching out-degree bounds to their respective amortized solutions.

Keywords Dynamic graph algorithms · graph arboricity · edge orientations

1 Introduction

An important building block in algorithmic theory and practice is the ability to store graphs with low memory usage and fast query times. Classical storage methods are edge lists and adjacency matrix, but both have pitfalls for *sparse* graphs: adjacency matrices use too much memory, while edge lists can have slow adjacency queries and/or updates on high-out-degree vertices. Much research has been devoted to improving these simple methods. The graph parameter *arboricity* α is a well-known measure of a graph's sparsity, which captures the minimum number of forests the edges of a graph can be partitioned into. Kannan et al. [7] showed how to efficiently store static graphs with low arboricity and supporting fast ($\mathcal{O}(\alpha)$ time) adjacency queries in the worst case.

Brodal and Fagerberg [4] extended this idea to *dynamic* graphs, where edges may be arbitrarily inserted or deleted. If the arboricity of the graphs remains bounded by a constant α, the forest partitions may be forced to change due to the updates. The authors deal with this by considering the problem of orienting the edges of the dynamic graph as in [7], but by re-orienting ("flipping") edges as needed to maintain low out-degree. They gave a simple greedy algorithm and proved that its amortized number of flips was $\mathcal{O}(1)$-competitive to the number of flips made by any other sequence of flips that maintains a bounded out-degree. In this paper, we will use the term *(offline) strategy* to describe such a sequence. In particular, Brodal and Fagerberg showed how to maintain the out-degrees bounded by $\mathcal{O}(\alpha)$ with $\mathcal{O}(\log n)$ amortized flips, where n is the number of vertices in the graph. They also gave a lower bound of $\Omega(n)$ flips for maintaining the out-degrees bounded by α. It is not hard to see that this bound holds even for $\alpha = 1$.

E. Berglin
Aarhus University E-mail: berglin@cs.au.dk

G. S. Brodal
Aarhus University E-mail: gerth@cs.au.dk

Table 1 Previous and new results for the dynamic edge orientation of dynamic graphs with bounded arboricity α. Flip bounds marked 'am.' are amortized per update, others are worst-case.

Reference	Out-degree	Flips	α known	Note
Brodal & Fagerberg [4]	$\mathcal{O}(\alpha)$	$\mathcal{O}(\log n)$ am.	yes	$\Omega(n)$ worst-case flips
Kowalik [9]	$\mathcal{O}(\alpha \log n)$	$\mathcal{O}(1)$ am.	yes	uses alg. from [4]
Kopelowitz et al. [8]	$\mathcal{O}(\alpha + \log n)$	$\mathcal{O}(\alpha + \log n)$	no	
Kopelowitz et al. [8]	$\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$	$\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$	no	if $\alpha = \mathcal{O}\left(\sqrt{\log n}\right)$
He et al. [6]	$\mathcal{O}\left(\alpha\sqrt{\log n}\right)$	$\mathcal{O}\left(\sqrt{\log n}\right)$ am.	yes	uses alg. from [4]
He et al. [6]	$\mathcal{O}(\alpha \log n)$	$\mathcal{O}(\alpha \log n)$	no	
New (Corollary 3)	$\mathcal{O}(\alpha + \log n)$	$\mathcal{O}(\log n)$	no	
New (Corollary 4)	$\mathcal{O}(\alpha \log n)$	$\mathcal{O}\left(\sqrt{\log n}\right)$	no	
New (Corollary 5)	$\mathcal{O}(\log n)$	$\mathcal{O}\left(\alpha\sqrt{\log n}\right)$	yes	if $\alpha = \mathcal{O}\left(\sqrt{\log n}\right)$
New (Corollary 2)	$\mathcal{O}(\alpha \log^2 n)$	$\mathcal{O}(1)$	no	
New (Corollary 1)	$\mathcal{O}\left(\frac{\alpha \log^2 n}{f(n)}\right)$	$\mathcal{O}(f(n))$	no	if $f(n) = \mathcal{O}(\log n)$

Kowalik [9] gave another offline strategy and applied it to the algorithm by Brodal and Fagerberg, getting $\mathcal{O}(\alpha \log n)$ out-degree in constant amortized flips, demonstrating that a reasonable trade-off was possible. As stated, both the algorithms of Brodal and Fagerberg [4] and Kowalik [9] need to know, and use as a parameter, a bound on the arboricity of the graph. However it is straightforward to modify these algorithms to guess the arboricity, and automatically adjust the guess if the guess was wrong, without impacting their amortized time complexities.

Kopelowitz, Krauthgamer, Porat and Solomon [8] later found a different algorithm, which came with slightly worse bounds but in the worst case rather than amortized. Their algorithm maintains $\mathcal{O}(\alpha + \log n)$ out-degree with $\mathcal{O}(\alpha + \log n)$ flips, without knowing α. However, if α is known, they give an alternate algorithm with somewhat better running time for edge insertions, but otherwise equal bounds for out-degree and number of flips. Also, if $\alpha = \mathcal{O}\left(\sqrt{\log n}\right)$, both bounds can be improved slightly to $\mathcal{O}(\log n / \log\log n)$ due to some freedom in setting the base of the logarithmic terms.

He, Tang and Zeh [6] gave a new offline strategy with a parametrized trade-off between out-degree and flips, generalizing the two strategies in [4] and [9]. When applied to the algorithm by Brodal and Fagerberg it achieves $\mathcal{O}\left(\alpha\sqrt{\log n}\right)$ out-degree with $\mathcal{O}\left(\sqrt{\log n}\right)$ amortized flips. They also give another algorithm with worst-case bounds, nearly matching the out-degree and flips of [8] but with somewhat simpler pseudocode and (if $\alpha = \omega(\log n)$) improved time complexity for insertions.

The problem was originally motivated by quick adjacency queries [7]. But rather than making an explicit dictionary data structure, for most of the paper we focus on the problem of dynamically flipping edges to guarantee low maximum out-degree. This allows us to ignore lower bounds for dictionary operations, and we deliberately omit comparisons of update time complexity as they might be skewed unfairly in our favor. It is straightforward to create such a data structure on top of our machinery, should one so desire, by extending our solution to report which edges are flipped. This allows programmers to tailor the balance between update and query time to suit their own needs. In Section 7 we describe a higher-level data structure for adjacency queries with the best bounds yet (insertions, deletions and queries all in $\mathcal{O}(\log\log\log n)$ worst-case deterministic time) for a certain setting, as well as some updated results on maintaining an approximate maximum matching.

1.1 Our contribution

We present a new algorithm for maintaining an edge orientation of a dynamic graph, with a guarantee of low out-degree and worst-case number of flips. Like many previous solutions we relate the performance to the arboricity α of the dynamic graph, but unlike some previous works, ours does not require knowledge of the arboricity in the general case. Our algorithm is furthermore much simpler than previous ones, and uses queues as the only under-the-hood data structure. It owes its simplicity to the fact that it greedily chooses which edge to flip.

By controlling a run-time parameter, our algorithm allows a user-specified trade-off between the out-degree and the number of flips; this was previously only possible for algorithms with *amortized* number of flips. Depending on the choice of the parameter, the algorithm can maintain e.g. $\mathcal{O}(\alpha + \log n)$ out-degree with $\mathcal{O}(\log n)$ flips, or $\mathcal{O}\big(\alpha \log^2 n\big)$ out-degree with constant flips. Various other parameter settings are possible. We match or improve all known bounds with worst-case flips, except when the arboricity is within a specific, very narrow range.

2 Preliminaries

The *arboricity* of a graph G is the smallest number t such that the edges of G can be partitioned into t forests. Several equivalent definitions are used throughout the literature. We use arboricity(G) to denote the arboricity of G. A graph G with bounded arboricity arboricity$(G) \leq \alpha$ is *sparse*: any induced subgraph of G on $n' \leq n$ vertices contains at most $(n'-1)\alpha$ edges. Note that while bounded arboricity graphs have no dense neighbourhood they can still have vertices of arbitrarily high degree, e.g. stars have arboricity 1 but maximum degree $n-1$.

We say that $\mathcal{G} = G_0, G_1, G_2, \ldots, G_t$ is an *edit-sequence of graphs* if for each $i > 0$ there exists some edge (u,v) s.t. either $G_i = G_{i-1} \cup \{(u,v)\}$ (update i is an *insertion*) or $G_i = G_{i-1} \setminus \{(u,v)\}$ (update i is a *deletion*). We typically assume $G_0 = \emptyset$. We say that $\mathcal{G}$ has bounded arboricity (by a number α), or that arboricity$(\mathcal{G}) \leq \alpha$, if arboricity$(G_i) \leq \alpha$ for every i.

An *orientation* of a graph G is a directed graph $\overline{G}$ with the same vertex and edge sets as G, but where an undirected edge $(u,v) \in G$ exists as the directed edge (u,v) or (v,u) in $\overline{G}$. We use deg$(\overline{G})$ to denote the maximum out-degree of $\overline{G}$; it is a c-orientation if deg$(\overline{G}) \leq c$. Any graph G with arboricity$(G) \leq \alpha$ has an α-orientation; to see this, partition the edges into α forests, pick an arbitrary root in every tree, and direct every edge towards the root of its respective tree.

We say that $\overline{\mathcal{G}} = \overline{G}_0, \overline{G}_1, \ldots, \overline{G}_t$ is a *sequence of orientations* of $\mathcal{G}$ if every $\overline{G}_i$ is an orientation of G_i. Similarly, $\overline{\mathcal{G}}$ is a c-orientation if every $\overline{G}_i$ is a c-orientation. A *flip* is a triple (i,v,u) such that (v,u) is an edge in $\overline{G}_{i-1}$ and (u,v) is an edge in $\overline{G}_i$.

An *online c-orientation algorithm* $\mathcal{A}$ receives $\mathcal{G}$ as a stream and upon receiving *update* i, it produces $\overline{G}_i$ as a function of G_i and $\overline{G}_{i-1}$. We also say that $\mathcal{A}$ *maintains* an online c-orientation of $\mathcal{G}$.

To distinguish from an online algorithm, an *(offline) c-orientation strategy κ* is a sequence of c-orientations of $\mathcal{G}$. A strategy κ is not necessarily computable, but when it is, we will abuse notation and also let the *strategy κ* refer to some algorithm producing κ.

We say that κ or $\mathcal{A}$ makes σ flips (in the worst case) if the number of flips between any two updates $i, i+1$ is at most σ, and that it makes σ *amortized* flips if after any update i the total amount of flips is at most σi.

3 The algorithm

The algorithm takes an edit-sequence of graphs $\mathcal{G}$ as an online stream, and a positive integer parameter k. Each vertex v maintains a standard FIFO queue Q_v which holds all of its out-edges; by deg(v) we mean $|Q_v|$. On an insertion (deletion) update, orient the new edge arbitrarily (delete the edge via object reference) and then k times pick a vertex v with maximum out-degree and flip the first edge in Q_v.

The book-keeping of out-degrees is easy to accomplish with $\mathcal{O}(1)$ overhead. Use e.g. an array A where each $A[i]$ holds a doubly-linked list of every vertex with out-degree i, plus a pointer to the highest index in A containing a non-empty list. The pointer is easy to keep updated since insertions, deletions and flips can only change the out-degree of a vertex by 1.

We do not explicitly support queries. See pseudocode below for ease of reading.

Algorithm 1 Greedy flipping algorithm

```
procedure INSERTION($(v, u)$)
    push $(v, u)$ to $Q_v$
    K-FLIPS
procedure DELETION($(v, u)$)
    remove $(v, u)$ from $Q_v$
    K-FLIPS
procedure K-FLIPS
    for $i = 1$ to $k$ do
        let $v$ be a max out-degree vertex
        pop an edge $(v, u)$ from $Q_v$
        push $(u, v)$ to $Q_u$
```

Table 2 Edge potential by type and placement in queue.

	Front	Back
Good	$1 + 2\varepsilon$	$1 - \varepsilon$
Bad (first 3δ)	1	$1 + \varepsilon$
Bad (rest)	1	1

4 Analysis

To show the efficiency of Algorithm 1, we will prove that its out-degree is competitive to an unknown offline strategy. For given $\mathcal{G}$ and k, let δ, σ and ε be values satisfying the following conditions: (i) there exists an offline δ-orientation strategy κ of $\mathcal{G}$ making at most σ flips in the worst case, (ii) $0 < \varepsilon \leq 1$, and (iii) $k \geq 1 + 1/\varepsilon + 2\sigma$.

Remark 1 Algorithm 1 is completely oblivious to the values of δ, σ and ε, as well as any graph properties of $\mathcal{G}$ itself. Consequently it is possible to perform the analysis against several different strategies simultaneously, including strategies for only a contiguous sub-sequence of $\mathcal{G}$.

Theorem 1 *Algorithm 1 maintains an $\mathcal{O}(\delta + (\delta\varepsilon + 1)\log_2 n)$-orientation of $\mathcal{G}$ with k flips and in $\mathcal{O}(k)$ time.*

The number of flips, and hence the running time, in Theorem 1 is trivial from the pseudocode. The rest of this section is dedicated to proving the bound on the out-degree. While the proof is quite non-trivial, the roadmap thereof is easy. We will associate potentials on all edges, so that the potential of an edge depends on where it is stored – hence potential is inserted or removed from the system as the edges are moved around. Then we show that either the total potential cannot increase, or the maximum out-degree is $\mathcal{O}(\delta)$ in which case the potentials do not matter. Finally we re-interpret the moving of potentials as a game, where even an adversary cannot concentrate more than $\mathcal{O}((\delta\varepsilon + 1)\log n)$ extra potential in any single vertex – this also (roughly) bounds the maximum out-degree.

For purposes of analysis, we consider each queue Q_v to be two queues, the *Front* F_v and *Back* B_v. Edges are always inserted into B_v, and extracted from F_v. If F_v is empty when an edge should be extracted from Q_v, simply swap the two queues (by renaming) and then continue. Clearly, this is equivalent to using a single queue. We say an edge was flipped *from* v and *to* u if it was removed from Q_v/F_v and inserted into Q_u/B_u.

To bound the maximum out-degree, we introduce potentials on the edges. At update i, we say that an edge in $\overline{G}_i$ is *good* if it has the same orientation as in $\kappa(G_i)$ and *bad* otherwise. Good edges have $1 + 2\varepsilon$ potential if they are in a Front queue and $1 - \varepsilon$ in a Back queue. Bad edges have potential 1, except for the first 3δ bad edges in any Back queue which have potential $1 + \varepsilon$. See Table 2 for an overview.

Let $p(v)$ be the sum of potentials of all edges stored in Q_v, $\hat{p}(\overline{G}) = \max_v p(v)$ and $P(\overline{G}) = \sum_v p(v)$. When we need to differentiate the potential of a vertex in a specific orientation $\overline{G}_i$, we use $p_i(v)$ to denote $p(v)$ at the time that the algorithm was storing $\overline{G}_i$.

Since Algorithm 1 does not know the values of δ or ε, it cannot determine the exact potential of a vertex. But as the following lemma shows, the out-degree of a vertex is a close approximation of its potential. We will prove the theorem by bounding the maximum potential of any vertex, which then implies a bound on its out-degree.

Lemma 1 *For any vertex v,* $\deg(v) + 5\delta\varepsilon \geq p(v) \geq \deg(v) - \delta\varepsilon$.

Proof For the upper bound, all edges contribute a base 1 potential, accounting for the $\deg(v)$ term. Note that at most δ out-edges of v are good. If they are all placed in F_v, they contribute an extra $2\delta\varepsilon$. At most 3δ bad edges in B_v contribute an extra ε each, giving at most $5\delta\varepsilon$ extra potential in total.

For the lower bound, only good edges in B_v can contribute less than 1 potential. Again there are at most δ of these and they contribute ε less, giving at least $\deg(v) - \delta\varepsilon$ potential in the vertex. □

Let $\beta = 6\delta\varepsilon$ be the *resolution* of the system. The following states that the potential of the maximum out-degree vertex is not too far from the maximum potential of any vertex.

Lemma 2 *Let u be some vertex with maximum potential, and let v be some maximum out-degree vertex. Then $p(u) - p(v) \leq \beta$.*

Proof By Lemma 1, the potential of v is at least $p(v) \geq \deg(v) - \delta\varepsilon$ and the potential of u is at most $p(u) \leq \deg(u) + 5\delta\varepsilon \leq \deg(v) + 5\delta\varepsilon$. Rearranging we get $p(u) - p(v) \leq \deg(v) + 5\delta\varepsilon - (\deg(v) - \delta\varepsilon) = 6\delta\varepsilon = \beta$. □

The next two lemmas are used to show that the potential function is 'well-behaved' when performing flips from vertices with high enough out-degree.

Lemma 3 *Assume a vertex v has an empty F_v and at least 4δ edges in B_v. Then swapping F_v and B_v does not increase $p(v)$.*

Proof The Back queue contains at most δ good edges and at least 3δ bad edges; hence exactly 3δ bad edges carry an extra ε potential which is released when moving from B_v to F_v. This $3\delta\varepsilon$ potential is enough to raise the potential of all δ good edges from $1 - \varepsilon$ to $1 + 2\varepsilon$. □

In the case that there are fewer than δ good edges present in the queue at the moment of swapping F_v and B_v, any surplus potential released from bad edges is simply lost.

Lemma 4 *Let v have out-degree at least 4δ. Then flipping an edge from v releases at least ε potential.*

Proof By Lemma 3 we can assume F_v is non-empty. Let (u, v) be the edge moved from F_v to B_u. Note that if the edge was previously good it is now bad, and vice versa. Hence its potential decreases either from $1 + 2\varepsilon$ to at most $1 + \varepsilon$, or from 1 to $1 - \varepsilon$. □

We now shift our focus away from the potentials, and consider the out-degrees after an uninterrupted sequence of flips. Lemma 5 upper bounds the maximum out-degree after the flips. Specifically, if a prefix of the sequence lowers the maximum out-degree to some value, then the rest of the sequence (almost) cannot raise the out-degree again.

Lemma 5 *Let S be any suffix of the sequence of flips performed by Algorithm 1 after some update. Let $d = \deg(\overline{G})$ at the start of S. Then $\deg(\overline{G}) \leq d + 1$ after S.*

Proof Note that flips can increase the maximum out-degree only if there are at least two vertices u, v with maximum out-degree, and the algorithm flips an edge incident on both of them. As soon as some vertex reaches out-degree $d + 1$, it will be the only vertex of maximum out-degree and immediately fall down to out-degree d in the following flip. Consequently no sequence of flips can raise a second vertex to out-degree $d + 1$, which is a necessary condition for raising any vertex to out-degree $d + 2$. □

Hence, to bound the out-degree after *all k* flips, we only need to bound the out-degree before some individual flip anywhere in the sequence.

Conversely, Lemma 6 *lower* bounds the out-degrees of any vertex that loses edges via flipping, in relation to the rest of the graph. That is to say, vertex that used to have maximum out-degree will, after all flips, still have *close to* maximum out-degree. Later when we introduce the game, this lemma will be a crucial component.

Lemma 6 *Let v be a vertex that had an edge flipped from it on update i. Then* $\deg_{\overline{G}_i}(v) \geq \deg(\overline{G}_i) - 2$.

Proof Take the suffix S of flips that begins with the last flip from v. Before S, $\deg(v) = d$ which is the maximum out-degree in $\overline{G}$. After S, $d-1 \leq \deg_{\overline{G}_i}(v)$ and $\deg(\overline{G}_i) \leq d+1$ by Lemma 5. □

Consider the algorithm as it receives an update i. We say that the currently stored graph $\overline{G}_{i-1}$ has *sufficient degree* if each of the k flips associated with update i happens from a vertex with out-degree at least 4δ. Conversely, we say the graph $\overline{G}_{i-1}$ has *insufficient degree* if at least one of the k flips is from a vertex with out-degree less than 4δ.

Recall that bounding the out-degree of the graph is the overall goal. The following two lemmas show that *either* the total potential stored in the system is a non-increasing quantity, *or* the graph has nicely bounded out-degree anyway. In the latter case the potentials are of no interest.

Lemma 7 *If $\overline{G}_{i-1}$ has insufficient degree, then* $\deg(\overline{G}_i) = \mathcal{O}(\delta)$.

Proof Since some edge was flipped from a vertex with out-degree $d < 4\delta$, it follows from Lemma 5 that $\deg(\overline{G}_i) \leq d + 1 \leq 4\delta$. □

Lemma 8 *If $\overline{G}_{i-1}$ has sufficient degree, then on update i the sum of potential released by flipping is at least as large as the sum of inserted potential.*

Proof Since $\overline{G}_{i-1}$ has sufficient degree, each flip frees ε potential by Lemma 4 and together they release $k\varepsilon$.

The offline strategy κ makes at most σ flips, which causes σ stored edges to swap their classification ("renaming") from good to bad or vice versa. A Front edge that was bad increases potential from 1 to $1+2\varepsilon$, and a Back edge that was good increases from $1-\varepsilon$ to at most $1+\varepsilon$. The renaming can therefore increase the total potential by at most $2\sigma\varepsilon$. Assuming update i is an insertion, the new edge is inserted into a Back queue, and adds at most $1+\varepsilon$ potential, and the sum of inserted potential is at most $1+\varepsilon+2\sigma\varepsilon$. If the update was instead a deletion, at most $2\sigma\varepsilon$ potential is inserted.

The lemma therefore holds as long as $k\varepsilon \geq 1+\varepsilon+2\sigma\varepsilon$, which is guaranteed by the choice of these parameters. □

Remark 2 If κ is *known* not to perform any flips on deletion updates, *no* potential gets added to the system (refer to the proof of Lemma 8). Hence our algorithm can also forgo flipping on deletions, without risking that the total potential increases.

Note that the potential of the system can increase on both insertion and deletion updates if the graph has insufficient degree, since we cannot rely on Lemma 3 to ensure that the potential of a vertex is well-behaved when flipping edges from it.

So far we have shown that either the maximum out-degree is $\mathcal{O}(\delta)$, or we have a non-increasing quantity of potential and the out-degree of each vertex is closely approximated by its own potential. We next bound the maximum out-degree via a *counter game*, disassociated from the actual graph orientation, played by an adversary whose goal it is to concentrate as much potential as possible in a single counter. Counter games have been explored previously, under various names, in e.g. [5] and [10]: typically they may be thought of as two-player games where the second player is benevolent. Our game is different because the lone player is instead restricted by the concept of resolution β.

Formally, the game is played by a single player on n counters $x_1, \ldots, x_n$. Each counter x_i will hold a non-negative real-valued *weight* $|x_i|$, and the sum of weights is a constant $\sum_i |x_i| = X$. Any such distribution of X on the n counters is called a *game configuration* C. Let $\hat{x} = \max_i |x_i|$ be the maximum weight at any time. The player can perform arbitrarily many iterations of the following three-step operation: (i) pick a counter x_i and a $c > 0$ such that $|x_i| - c \geq \max(0, \hat{x} - \beta - 2)$, (ii) remove c weight from x_i and (iii) add positive weights whose sum is c to any set of counters.

The player is therefore allowed to redistribute weight *to* arbitrary counters, but must take it in not-too-large chunks, and only *from* counters that are within the resolution (here $\beta + 2$) of the maximum counter. Before upper-bounding $\hat{x}$, we show that the player is powerful enough to simulate the movement of potentials by Algorithm 1. We say a game configuration C *dominates* a graph orientation $\overline{G}$ if $|x_j| \geq p(v_j)$ for every j.

Lemma 9 *Let i be an update such that $\overline{G}_{i-1}$ has sufficient degree. Let C be a game configuration that dominates $\overline{G}_{i-1}$. Then the player can reach a game configuration C' that dominates $\overline{G}_i$.*

Proof We need to show that if some vertex gains potential (so its corresponding counter no longer dominates it), then we can safely take enough weight from other counters to fill that 'gap'. Keep in mind that the total potential does not increase (Lemma 8). Since the player is allowed to redistribute weight *to* any counter, we let the gaps be filled in arbitrary order and only show that enough weight can be taken *from* other counters to make up the difference. If $\hat{x} > \hat{p}(\overline{G}_{i-1})$ then greedily take weight from all counters greater than $\hat{p}(\overline{G}_{i-1})$ to get $\hat{x} = \hat{p}(\overline{G}_{i-1})$. This first step is required mostly out of technicality: we *want* to access the counters with at least $\hat{p}(\overline{G}_{i-1}) - \beta - 2$ weight, but *can* only access those with at least $\hat{x} - \beta - 2$ weight.

Let v_j be a vertex that had an edge flipped from it. Then its resulting out-degree is $\deg_{\overline{G}_i}(v_j) \geq \deg(\overline{G}_i) - 2$ (Lemma 6) and its potential is $p_i(v_j) \geq \deg_{\overline{G}_i}(v_j) - \delta\varepsilon \geq \deg(\overline{G}_i) - 2 - \delta\varepsilon$ (Lemma 1). Also by Lemma 1 the maximum potential in the system is $\hat{p}(\overline{G}_i) \leq \deg(\overline{G}_i) + 5\delta\varepsilon$. Hence the final potential of v_j is within $6\delta\varepsilon + 2 = \beta + 2$ of the maximum potential. As the rules of the counter game allow us to take weight up to $\beta + 2$ from the maximum counter, then however much potential v_j lost we can take at least the same amount of weight from its corresponding counter x_j.

Conversely, if a vertex loses potential but its resulting potential is not at least $\hat{p}(\overline{G}_i) - \beta - 2$, it must have lost that potential due to deletion or renaming rather than flipping. Its counter can safely be left untouched and still dominate the potential of the vertex.

Since the sum of potential decreases (by flipping) is at least as large as the sum of increases (for any reason) (Lemma 8), and for any vertex that lost potential by flipping we can remove at least as much weight from its counter, then we can redistribute enough weight to raise the too-low counters to again dominate their respective vertex potentials. The updated counters form a game configuration that dominates $\overline{G}_i$. □

Lemma 10 *Let $\overline{G}_a, \ldots, \overline{G}_b$ be any sequence of orientations such that $\overline{G}_i$ has sufficient degree for every $a \leq i \leq b$. Consider a game with starting configuration C_a that dominates $\overline{G}_a$, with $\hat{x} = \hat{p}(\overline{G}_a)$. Then the player can reach game configurations $C_a, \ldots, C_b$ where C_i dominates $\overline{G}_i$ for every $a \leq i \leq b$.*

Proof For every $a < i \leq b$ iterate Lemma 9 on C_{i-1} to create C_i. □

We now let an adversary play the game, with the goal to increase $\hat{x}$ as much as possible. For simplicity we assume that every counter is raised to $\hat{x}$ to form the starting configuration. For $j = -1, 0, 1, 2, \ldots$ let $\ell_j = X/n + j(\beta + 2)$ be *weight level j*. A counter x_i is *above level j*, or *above ℓ_j*, if $|x_i| \geq \ell_j$. Let $X_j = \sum_{i=1}^{n} \max(0, |x_i| - \ell_j)$ be the *weight above ℓ_j*, and $\overline{X}_j = X - X_j$ the *weight below ℓ_j*. We say a counter x_i *contributes* $\max(0, |x_i| - \ell_j)$ to X_j and $\min(|x_i|, \ell_j)$ to $\overline{X}_j$.

Lemma 11 *Let j be a weight level such that $\ell_j \leq \hat{x}$. Let the player make any sequence of moves that maintain the condition $\ell_j \leq \hat{x}$. Then X_{j-1} does not increase.*

Proof Note that any counter x_i contributes $\min(|x_i|, \ell_{j-1})$ to $\overline{X}_{j-1}$. By assumption there will always be a counter x_k with $\ell_j \leq |x_k|$. Hence the resolution rule prevents the player from making any counter contribute less to $\overline{X}_{j-1}$ than it already does. Since X is a constant and $\overline{X}_{j-1}$ is non-decreasing, $X_{j-1} = X - \overline{X}_{j-1}$ is non-increasing. □

Lemma 12 *Let j be a weight level such that $\ell_j \leq \hat{x} \leq \ell_{j+1}$. Let the player make any sequence of moves that maintain the condition $\ell_j \leq \hat{x} \leq \ell_{j+1}$. Then $2X_j \leq X_{j-1}$.*

Proof By Lemma 11, X_{j-1} is a non-increasing amount. Let x_i be any counter that will contribute some positive weight w to X_j. Since the player maintains that $\hat{x} \leq \ell_{j+1}$, no counter will be able to contribute more than $\ell_{j+1} - \ell_j = \beta + 2$ to X_j, i.e. $0 < w \leq \beta + 2$. Then x_i must contribute $w + \beta + 2$ to X_{j-1}. Hence any counter that contributes to X_j contributes at least twice as much to X_{j-1}, and $2X_j \leq X_{j-1}$. □

The player is therefore stuck in the following dilemma: once $\hat{x}$ reaches some level ℓ_j, only a bounded amount X_{j-1} of weight remains available to redistribute, due to Lemma 11. But once $\hat{x}$ reaches ℓ_{j+1}, the weight below ℓ_j can no longer be redistributed due to the resolution rule. Therefore, in order to concentrate as much weight as possible above ℓ_{j+1}, the player must first maximize X_j without any counter actually reaching above ℓ_{j+1}.

Lemma 13 *The player cannot increase $\hat{x}$ to $\ell_{1+\log_2 n}$.*

Proof Assume $\hat{x} \geq \ell_{\log_2 n}$. Since $\ell_0 = X/n$ is the average weight of all counters, it must always be the case that $\hat{x} \geq \ell_0$ and $X_{-1} \leq n(\beta + 2)$. By alternatingly iterating Lemma 11 and Lemma 12, the weight above $\ell_{\log_2 n}$ is $X_{\log_2 n} \leq \left(\frac{1}{2}\right)^{1+\log_2 n} X_{-1} = \frac{1}{2n} X_{-1} \leq \frac{1}{2n} n(\beta + 2) < \beta + 2$. Since the weight is strictly less than $\beta + 2$, even concentrating all of it in a single counter is not enough to make that counter reach $\ell_{1+\log_2 n}$. Hence $\hat{x} < \ell_{1+\log_2 n}$. □

We are now ready to prove the out-degree part of Theorem 1.

Proof (Theorem 1) Either the graph orientation has insufficient degree and maximum out-degree $\mathcal{O}(\delta)$ (Lemma 7), or it has non-increasing potential (Lemma 8) which is dominated by a counter game (Lemma 10) where the starting weight of any counter is at most $4\delta + 5\varepsilon\delta = \mathcal{O}(\delta)$. By Lemma 13, the maximum counter is $\hat{x} < \ell_{1+\log_2 n} = \mathcal{O}(\delta) + (1 + \log_2 n)(\beta + 2)$. By Lemma 1, $\deg(v) \leq p(v) + \delta\varepsilon$, and therefore any vertex has out-degree bounded by $\mathcal{O}(\delta) + (1 + \log_2 n)(\beta + 2) + \delta\varepsilon = \mathcal{O}(\delta + (\delta\varepsilon + 1)\log_2 n)$. □

5 De-amortizing offline strategies

In the previous work by Brodal and Fagerberg [4], their amortized algorithm is shown competitive with an offline strategy that has bounded amortized number of flips, and hence subsequently published strategies have focused on achieving good amortized bounds. However, for our algorithm analysis, we require an offline strategy with *worst-case* flips per update. In this section we show one way to de-amortize offline strategies. Our technique does not generalize to every offline strategy, but relies on the special structure inherent to the strategies of both [8] and [6]. These strategies partition the edit-sequence into blocks of consecutive updates, with some length λ. No flips occur within a block, only in the seams between two blocks. The amortized flip complexity of these strategies is therefore simply the maximum number of flips between two blocks, divided by the length λ of the preceding block.

Since no flips are allowed within a block, the strategy is required to find an orientation of the union of all graphs $G_i, \ldots, G_{i+\lambda-1}$ within a block. The maximum out-degree of the entire strategy is therefore upper bounded by the maximum out-degree of any oriented union-graph. Higher λ gives a less sparse union-graph, necessitating higher out-degree, but also allows for a better amortized flip complexity. The following theorem shows a simple way of de-amortizing strategies with this structure, by taking all the flips between two blocks and spreading them evenly over the updates in the later block.

Theorem 2 *Let κ be a δ-orientation strategy of $\mathcal{G}$ where, for arbitrary λ, any update with $\sigma\lambda$ flips is followed by at least $\lambda - 1$ updates with no flips. Then there exists a 2δ-orientation strategy of $\mathcal{G}$ making σ flips in the worst case.*

Note that if the last block of flips is not followed by $\lambda - 1$ updates due to $\mathcal{G}$ ending, then one can pad $\mathcal{G}$ to appropriate length by repeatedly inserting and removing a dummy edge after the end of $\mathcal{G}$. Also note that λ can vary within the same sequence – blocks do not need to be of uniform length.

Proof Let i be an update where κ performs a set of $\lambda\sigma$ flips. Let F be the set of flipped edges. Let κ' be an offline strategy with the same edge orientations as κ except for updates $i, \dots, i+\lambda-1$. On any insertion update $i, \dots, i+\lambda-1$, let κ' orient the new edge in the same direction as κ. Furthermore, on each update $i, \dots, i+\lambda-1$, κ' takes σ arbitrary edges in F, removes them from F, and flips them.

Then F will be empty after update $i+\lambda-1$, so $\kappa(G_{i+\lambda-1}) = \kappa'(G_{i+\lambda-1})$. At all times F forms a δ-orientation, since F is a subset of $\kappa(G_{i-1})$. Similarly, $\kappa'(G_j) \setminus F$ is δ-orientation for every $i \le j \le i+\lambda-1$, since they are a subset of $\kappa(G_j)$. Hence κ' is a 2δ-orientation. Finally, κ' performs at most σ flips per update between updates i and $i+\lambda-1$; exhaustively perform the same transformation on all of κ for σ flips on any update. □

6 Discussion

With our two theorems proven, we can relate the algorithm to known offline strategies and achieve the following corollaries. In all of the following, $\mathcal{G}$ is an arbitrary edit-sequence with arboricity$(\mathcal{G}) \le \alpha$.

Kowalik [9] presents an offline $\mathcal{O}(\alpha \log n)$-orientation strategy making 1 amortized flip. Using Theorem 2 we can de-amortize it to an offline $\mathcal{O}(\alpha \log n)$-orientation strategy making 1 flip in the worst case, giving the following two corollaries.

Corollary 1 *For a positive function $f(n) = \mathcal{O}(\log n)$, Algorithm 1 maintains an $\mathcal{O}\Big(\frac{\alpha \log^2 n}{f(n)}\Big)$-orientation with $k = 3 + \lceil f(n) \rceil$ flips.*

Proof Let $\delta = \mathcal{O}(\alpha \log n)$, $\sigma = 1$ and $\varepsilon = 1/f(n)$. Then the algorithm maintains out-degree $\mathcal{O}\Big(\alpha \log n + \Big(\frac{\alpha \log n}{f(n)} + 1\Big) \log n\Big) = \mathcal{O}\Big(\frac{\alpha \log^2 n}{f(n)}\Big)$. □

Corollary 2 *Algorithm 1 maintains an $\mathcal{O}\big(\alpha \log^2 n\big)$-orientation of $\mathcal{G}$ with $k = 4$ flips.*

Proof Let $f(n) \equiv 1$ in Corollary 1. □

Corollary 2 is the first result with $\mathcal{O}(1)$ worst-case flips. Compared to [9] (with $\mathcal{O}(1)$ *amortized* flips), it incurs an extra $\mathcal{O}(\log n)$ factor on the out-degree, but avoids the $\Omega(n)$ worst-case flips which that algorithm can experience.

Brodal and Fagerberg [4] give an offline $\mathcal{O}(\alpha)$-orientation strategy with $\mathcal{O}(\log n)$ flips in the worst case. It performs no flips on deletion updates, so Algorithm 1 can also forgo flips on deletions (Remark 2).

Corollary 3 *Algorithm 1 maintains an $\mathcal{O}(\alpha + \log n)$-orientation of $\mathcal{G}$ with $k = \mathcal{O}(\log n)$ flips.*

Proof Let $\delta = \mathcal{O}(\alpha)$, $\sigma = \mathcal{O}(\log n)$ and $\varepsilon = 1/\log n$. Then the algorithm maintains out-degree $\mathcal{O}\Big(\alpha + \Big(\frac{\alpha}{\log n} + 1\Big) \log n\Big) = \mathcal{O}(\alpha + \log n)$. □

He et al. [6] give an offline $\mathcal{O}\big(\alpha\sqrt{\log n}\big)$-orientation strategy making $\mathcal{O}\big(\sqrt{\log n}\big)$ amortized flips, which we de-amortize using Theorem 2.

Corollary 4 *Algorithm 1 maintains an $\mathcal{O}(\alpha \log n)$-orientation of $\mathcal{G}$ with $k = \Theta(\sqrt{\log n})$ flips.*

Proof Let $\delta = \mathcal{O}\big(\alpha\sqrt{\log n}\big)$, $\sigma = \mathcal{O}\big(\sqrt{\log n}\big)$ and $\varepsilon = 1/\sqrt{\log n}$. Then the algorithm maintains out-degree $\mathcal{O}\Big(\alpha\sqrt{\log n} + \Big(\frac{\alpha\sqrt{\log n}}{\sqrt{\log n}} + 1\Big) \log n\Big) = \mathcal{O}(\alpha \log n)$. □

Corollary 5 *Algorithm 1 maintains an $\mathcal{O}(\log n)$-orientation of $\mathcal{G}$ with $k = \mathcal{O}\big(\alpha\sqrt{\log n}\big)$ flips, if $\alpha = \mathcal{O}\big(\sqrt{\log n}\big)$.*

Proof Let $\delta = \mathcal{O}\big(\alpha\sqrt{\log n}\big)$, $\sigma = \mathcal{O}\big(\sqrt{\log n}\big)$ and $\varepsilon = 1/\alpha\sqrt{\log n}$. Then the algorithm maintains out-degree $\mathcal{O}\Big(\alpha\sqrt{\log n} + \Big(\frac{\alpha\sqrt{\log n}}{\alpha\sqrt{\log n}} + 1\Big) \log n\Big) = \mathcal{O}(\log n)$. □

Corollary 4 is an improvement over [8] in the flip complexity for edit-sequences with arboricity bounded by a constant. For $\alpha = \mathcal{O}\big(\sqrt{\log n}/\log\log n\big)$, Corollary 5 matches or improves the flip complexity from [8], albeit with a slightly worse out-degree bound, and only if α is known. If α is both $\mathcal{O}\big(\sqrt{\log n}\big)$ and $\omega(\sqrt{\log(n)}/\log\log n)$ we are narrowly outperformed by [8], by no more than an $\mathcal{O}(\log\log n)$ factor.

Corollary 4 also compares well against the amortized result from [6], with a $\mathcal{O}\big(\sqrt{\log n}\big)$ factor gap in the out-degree but with worst-case flips instead of amortized. Corollary 3 matches the bounds in [8] for general arboricity and improves on their flip complexity if $\alpha = \omega(\log n)$. Furthermore, if $\alpha = \Omega(\log n)$, Corollary 3 matches the amortized bounds from [4].

6.1 Reverse trade-off

Compared to an offline strategy, our analysis lends itself to a trade-off in one direction, getting (at most) an $\mathcal{O}(\log n)$ factor on the out-degree for a constant factor on the number of flips. It allows us to perform much fewer flips than in [8] at the price of weaker out-degree bounds. A trade-off in the opposite direction would also be highly desirable, achieving out-degree (closer to) $\mathcal{O}(\delta)$ by making $\Omega(\sigma)$ flips. We have only found a very weak such trade-off:

Lemma 14 *Algorithm 1 can maintain an $\mathcal{O}(\alpha)$-orientation of $\mathcal{G}$ with $k = \mathcal{O}(\alpha n)$ flips.*

Proof Let $\delta = \mathcal{O}(\alpha)$ and $\varepsilon = 1$ (the value of σ is inconsequential). Then each edge holds between 0 and 3 potential. And since any G_i has at most αn edges (by definition of arboricity), the total potential is between 0 and $3\alpha n$. Furthermore each flip releases 1 potential from the system, contingent on the graph having sufficient degree (Lemma 4). Hence after performing at most $3\alpha n$ flips on any starting orientation $\overline{G_i}$, we must reach a state where the next flip does not release potential, contradicting Lemma 4, and so by Lemma 7 the graph has out-degree at most $4\delta = \mathcal{O}(\alpha)$ after all flips. □

Lemma 14 only matches the worst-case bound of the algorithm in [4], which has drastically better amortized performance. Hence it should not be used in practice. Still, we believe a stronger reverse trade-off is possible and conjecture the following:

Conjecture 1 There exists a positive increasing function f such that for any edit-sequence $\mathcal{G}$ with a δ-orientation strategy making σ flips in the worst case, and for any $k \geq 2 + \sigma$, Algorithm 1 maintains an online $\mathcal{O}\Big(\delta + \frac{\sigma+1}{f(k)}\delta \log n\Big)$-orientation of $\mathcal{G}$ with k flips and in $\mathcal{O}(k)$ time.

6.2 Dynamic arboricity

Throughout the paper we have done all our performance analysis against a static arboricity bound, i.e. a bound on the greatest arboricity seen anywhere in the edit-sequence. An interesting issue arises if the sequence contains contiguous sub-sequences, of non-trivial length, with higher or lower arboricity than elsewhere in the sequence. Some previous algorithms, e.g. one of the algorithms in [8] and the non-amortized algorithm in [6], adapt to increasing and decreasing arboricity automatically.

Our analysis immediately adapts to sequences with increasing arboricity, since the analysis can be performed on any prefix (or contiguous sub-sequence) of $\mathcal{G}$ (see Remark 1). In the case of periods with lower arboricity than earlier in the sequence, our algorithm obeys the new arboricity if the maximum out-degree is already within that new bound. In other words, if the maximum out-degree is already bounded relative to the new arboricity, then it will remain so. However, if the arboricity falls enough that the current maximum out-degree *breaks* the new bounds, our analysis does not require the maximum out-degree to decrease accordingly. Intuitively, using a k strictly larger than $1 + 1/\varepsilon + 2\sigma$ (thus experiencing a net loss of total potential with every update) should force the maximum out-degree to tend towards the updated out-degree bounds, similar to the proof of Lemma 14. However, we do not have a formal argument for this.

6.3 Open problems

For all known strategies that maintain out-degree δ with σ (amortized) flips, it holds that $\delta\sigma = \Omega(\alpha \log n)$ and most achieve $\delta\sigma = \Theta(\alpha \log n)$. Can one design a strategy with $\delta\sigma = o(\alpha \log n)$? In [1] the first-named author shows that in the case of amortized flips, one needs only consider lazy strategies that flip a single directed path on insertion updates and do nothing on deletion updates.

7 Applications

7.1 Adjacency queries

Consider an application where an existing algorithm uses an orientation algorithm with out-degree Δ and f flips, as a black-box tool. A natural dimension of consideration, from the point of the higher-level algorithm, could be $\Delta + f$. None of our results improve on the previous solution with worst-case bounds in this dimension. But for an application whose time complexity depends only, say, logarithmically on Δ, we can leverage our lower flip complexity to yield improved bounds. One such application is adjacency queries in dynamic graphs where updates and queries are relatively equally frequent (and hence we choose to sacrifice query time in favor of improved update time).

Let $\mathcal{G}$ be an edit-sequence with arboricity$(\mathcal{G}) = \mathcal{O}(\log^t n)$ for some constant t. Kowalik [9] applies the orientation algorithm with $\Delta = \mathcal{O}(\alpha \log n) = \mathcal{O}(\log^{t+1} n)$ and $f = \mathcal{O}(1)$ amortized flips to perform insertions, deletions and queries in $\mathcal{O}(\log\log\log n)$ time but where the bound for insertions is amortized. By replacing their orientation algorithm with Corollary 2 (where $\Delta = \mathcal{O}(\alpha \log^2 n)$ and $f = \mathcal{O}(1)$ worst-case), we get the following.

Theorem 3 *Let G be a dynamic graph with* arboricity$(G) = \mathcal{O}(\log^t n)$ *at all times for some constant t. Then there exists a data structure which supports updates (insertion and deletion) and adjacency queries on G in $\mathcal{O}(\log\log\log n)$ worst-case time.*

This result improves on [9] by virtue of having only worst-case bounds. It also improves the worst-case solutions due to He et al. [6], with matching query time but where updates of both types can require $\Omega(\log n)$ time.

7.2 Maintaining a maximum matching

Starting with [11], orientation algorithms have been a popular black-box tool for maintaining approximate maximum matchings in dynamic graphs. The current state of the art with this approach is a $(3/2+\varepsilon)$-approximate maximum matching for arbitrary graph G [3], or $(1+\varepsilon)$-approximate if G is bipartite [2], in $\mathcal{O}(\alpha(\alpha + \log n))$ time for any constant $\varepsilon > 0$. Both results use the orientation algorithm from [8] as a subroutine, which performs edge insertions in $\mathcal{O}(\alpha(\alpha + \log n))$ time and this dominates the total running time to update the matching. Corollary 3 achieves essentially the same bounds on out-degree and flips as [8], but in only $\mathcal{O}(\log n)$ time. Hence by substituting their algorithm for ours we get the following.

Theorem 4 *Let G be a dynamic graph with* arboricity$(G) \leq \alpha$ *and fix a constant $\varepsilon > 0$. Then one can deterministically maintain a $(3/2 + \varepsilon)$-approximate maximum matching of G with worst-case time $\mathcal{O}(\alpha + \log n)$ per update. If G is bipartite, the approximation ratio is $(1 + \varepsilon)$.*

Note that after this substitution the time to update the matching is no longer dominated by the running time of the re-orientation algorithm (now $\mathcal{O}(\log n)$), but rather the maximum out-degree ($\mathcal{O}(\alpha + \log n)$) of the resulting orientation. However, both of these updated results have already been obsoleted by one due to Peleg and Solomon [12], which maintains an $(1 + \varepsilon)$-approximate matching in $\mathcal{O}(\alpha)$ time without the use of an orientation algorithm. Since the maximum out-degree is always at least $\alpha - 1$ in any orientation, it seems that any matching algorithm that employs an orientation algorithm will also require new ideas to push its time complexity to $o(\alpha)$.

References

1. Berglin, E.: Geometric covers, graph orientations, counter games. Ph.D. thesis, Aarhus University (2017)
2. Bernstein, A., Stein, C.: Fully dynamic matching in bipartite graphs. In: International Colloquium on Automata, Languages, and Programming, pp. 167–179. Springer (2015)
3. Bernstein, A., Stein, C.: Faster fully dynamic matchings with small approximation ratios. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 692–711. Society for Industrial and Applied Mathematics (2016)
4. Brodal, G.S., Fagerberg, R.: Dynamic representation of sparse graphs. In: Proceedings 6th International Workshop on Algorithms and Data Structures (WADS), *Lecture Notes in Computer Science*, vol. 1663, pp. 342–351. Springer (1999)
5. Dietz, P., Sleator, D.: Two algorithms for maintaining order in a list. In: Proceedings 19th Annual ACM Symposium on Theory of Computing (STOC), pp. 365–372. ACM (1987)
6. He, M., Tang, G., Zeh, N.: Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In: Proceedings 25th International Symposium on Algorithms and Computation (ISAAC), *Lecture Notes in Computer Science*, vol. 8889, pp. 128–140. Springer (2014)
7. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. SIAM Journal on Discrete Mathematics **5**(4), 596–603 (1992)
8. Kopelowitz, T., Krauthgamer, R., Porat, E., Solomon, S.: Orienting fully dynamic graphs with worst-case time bounds. In: Proceedings 41st International Colloquium Automata, Languages, and Programming (ICALP), Part II, *Lecture Notes in Computer Science*, vol. 8573, pp. 532–543. Springer (2014)
9. Kowalik, Ł.: Adjacency queries in dynamic sparse graphs. Information Processing Letters **102**(5), 191–195 (2007)
10. Levcopoulos, C., Overmars, M.H.: A balanced search tree with $O(1)$ worst-case update time. Acta Informatica **26**(3), 269–277 (1988)
11. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. ACM Transactions on Algorithms (TALG) **12**(1), 7 (2016)
12. Peleg, D., Solomon, S.: Dynamic $(1+\varepsilon)$-approximate matchings: a density-sensitive approach. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 712–729. Society for Industrial and Applied Mathematics (2016)