Improved Distributed Degree Splitting and Edge Coloring[*][†]

Mohsen Ghaffari[1], Juho Hirvonen[2], Fabian Kuhn[3], Yannic Maus[4], Jukka Suomela[5], and Jara Uitto[6]

1 **ETH Zürich, Switzerland**
`ghaffari@inf.ethz.ch`
2 **IRIF, CNRS, and University Paris Diderot, France**
`juho.hirvonen@irif.fr`
3 **University of Freiburg, Germany**
`kuhn@cs.uni-freiburg.de`
4 **University of Freiburg, Germany**
`yannic.maus@cs.uni-freiburg.de`
5 **Aalto University, Finland**
`jukka.suomela@aalto.fi`
6 **ETH Zürich, Switzerland, and University of Freiburg, Germany**
`jara.uitto@inf.ethz.ch`

Abstract

The degree splitting problem requires coloring the edges of a graph red or blue such that each node has almost the same number of edges in each color, up to a small additive discrepancy. The directed variant of the problem requires orienting the edges such that each node has almost the same number of incoming and outgoing edges, again up to a small additive discrepancy.

We present deterministic distributed algorithms for both variants, which improve on their counterparts presented by Ghaffari and Su [SODA'17]: our algorithms are significantly simpler and faster, and have a much smaller discrepancy. This also leads to a faster and simpler deterministic algorithm for $(2+o(1))\Delta$-edge-coloring, improving on that of Ghaffari and Su.

1998 ACM Subject Classification C.2.2 Network Protocols

Keywords and phrases Distributed Graph Algorithms, Degree Splitting, Edge Coloring, Discrepancy

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.19

1 Introduction and Related Work

In this work, we present improved distributed (LOCAL model) algorithms for the *degree splitting problem*, and also use them to provide simpler and faster deterministic distributed algorithms for the classic and well-studied problem of *edge coloring*.

LOCAL Model. In the standard LOCAL model of distributed computing[15, 17], the network is abstracted as an n-node undirected graph $G = (V, E)$, and each node is labeled with a unique $O(\log n)$-bit identifier. Communication happens in synchronous rounds of *message*

[*] Third, fourth and sixth author supported by ERC Grant No. 336495 (ACDC). Second author supported by Ulla Tuominen Foundation.

[†] A full version of the paper is available at `https://arxiv.org/abs/1706.04746`.

passing, where in each round each node can send a message to each of its neighbors. At the end of the algorithm each node should output its own part of the solution, e.g., the colors of its incident edges in the edge coloring problem. The time complexity of an algorithm is the number of synchronous rounds.

Degree Splitting Problems. The *undirected degree splitting* problem seeks a partitioning of the graph edges E into two parts so that the partition looks almost balanced around each node. Concretely, we should color each edge red or blue such that for each node, the difference between its number of red and blue edges is at most some small *discrepancy* value κ. In other words, we want an assignment $q\colon E \to \{+1, -1\}$ such that for each node $v \in V$, we have $\left|\sum_{e \in E(v)} q(e)\right| \leq \kappa$, where $E(v)$ denotes the edges incident on v. We want κ to be as small as possible.

In the *directed* variant of the degree splitting problem, we should orient all the edges such that for each node, the difference between its number of incoming and outgoing edges is at most a small discrepancy value κ.

Why Should One Care About Distributed Degree Splittings? On the one hand, degree splittings are natural tools for solving other problems with a *divide-and-conquer* approach. For instance, consider the well-studied problem of edge coloring, and suppose that we are able to solve degree splitting efficiently with discrepancy $\kappa = O(1)$. We can then compute an edge coloring with $(2+\varepsilon)\Delta$ colors, for any constant $\varepsilon > 0$; as usual, Δ is the maximum degree of the input graph $G = (V, E)$. For that, we recursively apply the degree splittings on G, each time reapplying it on each of the new colors, for a recursion of height $h = O(\log \varepsilon\Delta)$. This way we partition G in 2^h edge-disjoint graphs, each with maximum degree at most

$$\Delta' = \frac{\Delta}{2^h} + \sum_{i=1}^{h} \frac{\kappa}{2^i} \leq \frac{\Delta}{2^h} + \kappa = O(1/\varepsilon).$$

We can then edge color each of these graphs with $2\Delta' - 1$ colors, using standard algorithms (simultaneously in parallel for all graphs and with a separate color palette for each graph), hence obtaining an overall coloring for G with $2^h \cdot (2\Delta' - 1) \leq 2\Delta + 2^h\kappa = (2+\varepsilon)\Delta$ colors. We explain the details of this relation, and the particular edge coloring algorithm that we obtain using our degree splitting algorithm, later in Section 2.

On the other hand, degree splitting problems are interesting also on their own: they seem to be an elementary locally checkable labeling (LCL) problem[16], and yet, even on bounded degree graphs, their distributed complexity is highly non-trivial. In fact, they exhibit characteristics that are intrinsically different from those of the classic problems of the area, including maximal independent set, maximal matching, $\Delta + 1$ vertex coloring, and $2\Delta - 1$ edge coloring. All of these classic problems admit trivial sequential greedy algorithms, and they can also be solved very fast distributedly on bounded degree graphs, in $\Theta(\log^* n)$ rounds[15]. In contrast, degree splittings constitute a middle ground in the complexity: even on bounded degree graphs, deterministic degree splitting requires $\Omega(\log n)$ rounds, as shown by Chang et al. [6], and randomized degree splitting requires $\Omega(\log\log n)$ rounds, as shown by Brandt et al. [4]. These two lower bounds were presented for the *sinkless orientation* problem, introduced by Brandt et al. [4], which can be viewed as a very special case of directed degree splitting: In sinkless orientation, we should orient the edges so that each node of degree at least d, for some large enough constant d, has at least one outgoing edge. For this special case, both lower bounds are tight[11].

What is Known? First, we discuss the existence of low-discrepancy degree splittings. Any graph admits an undirected degree splitting with discrepancy at most 2. This is the best possible, as can be seen on a triangle. This low-discrepancy degree splitting can be viewed as a special case of a beautiful area called *discrepancy theory* (see e.g. [7] for a textbook coverage), which studies coloring the elements of a ground set red/blue so that each of a collection of given subsets has almost the same number of red and blue elements, up to a small additive discrepancy. For instance, by a seminal result of Beck and Fiala from 1981[2], any hypergraph of rank t (each hyperedge has at most t vertices) admits a red/blue edge coloring with per-node discrepancy at most $2t-2$. See [5, 3] for some slightly stronger bounds, for large t. In the case of standard graphs, where $t=2$, the existence proof is straightforward: Add a dummy vertex and connect it to all odd-degree vertices. Then, take an Eulerian tour, and color its edges red and blue in an alternating manner. In directed splitting, a discrepancy of $\kappa=1$ suffices, using the same Eulerian tour approach and orienting the edges along a traversal of this tour.

In the algorithmic world, Israeli and Shiloach [13] were the first to consider degree splittings. They used it to provide an efficient parallel (PRAM model) algorithm for maximal matching. This, and many other works in the PRAM model which later used degree splittings (e.g., [14]) relied on computing Eulerian tours, following the above scheme. Unfortunately, this idea cannot be used efficiently in the distributed setting, as an Eulerian tour is a non-local structure: finding and alternately coloring it needs $\Omega(n)$ rounds on a simple cycle.

Inspired by Israeli and Shiloach's method [13], Hanckowiak et al. [12] were the first to study degree splittings in distributed algorithms. They used it to present the breakthrough result of a polylog(n)-round deterministic distributed maximal matching, which was the first efficient deterministic algorithm for one of the classic problems. However, for that, they ended up having to relax the degree splitting problem in one crucial manner: they allowed a $\delta = 1/\operatorname{polylog} n$ fraction of nodes to have arbitrary splits, with no guarantee on their balance. As explained by Czygrinow et al. [8], this relaxation ends up being quite harmful for edge coloring; without fixing that issue, it seems that one can get at best an $O(\Delta \log n)$-edge coloring.

Very recently, Ghaffari and Su[11] presented solutions for degree splitting without sacrificing any nodes, and used this to obtain the first polylog n round algorithm for $(2+o(1))\Delta$-edge coloring, improving on prior polylog(n)-round algorithms that used more colors: the algorithm of Barenboim and Elkin [1] for $\Delta \cdot \exp(O(\frac{\log \Delta}{\log\log \Delta}))$ colors, and the algorithm of Czygrinow et al. [8] for $O(\Delta \log n)$ colors. The degree splitting algorithm of Ghaffari and Su[11] obtains a discrepancy $\kappa = \varepsilon\Delta$ in $O((\Delta^2 \log^5 n)/\varepsilon)$ rounds. Their method is based on iterations of flipping augmenting paths (somewhat similar in style to blocking flows in classic algorithms for the maximum flow problem[9]) but the process of deterministically and distributedly finding enough disjoint augmenting paths is quite complex. Furthermore, that part imposes a crucial limitation on the method: it cannot obtain a discrepancy better than $\Theta(\log n)$. As such, this algorithm does not provide any meaningful solution in graphs with degree $o(\log n)$.

Our Contributions. Our main result is a deterministic distributed algorithm for degree splitting that improves on the corresponding result of [11]. The new algorithm is (1) simpler, (2) faster, and (3) it gives a splitting with a much lower discrepancy.

▶ **Theorem 1.** *For every $\varepsilon > 0$, there are deterministic $O\big(\varepsilon^{-1}\cdot\log\varepsilon^{-1}\cdot\big(\log\log\varepsilon^{-1}\big)^{1.71}\cdot\log n\big)$-round distributed algorithms for computing directed and undirected degree splittings with the following properties:*

(a) *For directed degree splitting, the discrepancy at each node v of degree $d(v)$ is at most $\varepsilon \cdot d(v) + 1$ if $d(v)$ is odd and at most $\varepsilon \cdot d(v) + 2$ if $d(v)$ is even.*
(b) *For undirected degree splitting, the discrepancy at each node v of degree $d(v)$ is at most $\varepsilon \cdot d(v) + 4$.*

An important corollary of this splitting result is a faster and simpler algorithm for $(2 + o(1))\Delta$-edge coloring, which improves on the corresponding result from [11]. The related proof is deferred to the full version [10].

► **Corollary 2.** *For every $\varepsilon > 1/\log\Delta$, there is a deterministic distributed algorithm that computes a $(2+\varepsilon)\Delta$-edge coloring in $O\big(\log^2\Delta \cdot \varepsilon^{-1} \cdot \log\log\Delta \cdot (\log\log\log\Delta)^{1.71} \cdot \log n\big)$ rounds.*

This is significantly faster than the $O(\log^{11} n/\varepsilon^3)$-round algorithm of [11]. Furthermore, we are hopeful that with the future improvements in edge coloring for low-degree graphs, this splitting result will play an even more important role. Ideally, in the ultimate solution for edge coloring, say with $(1 + o(1))\Delta$ colors, this splitting will be one half of the solution: This half brings down the degree to a small value, with a negligible $(1 + o(1))$ factor loss, and the other half would hopefully color those small degree graphs efficiently.

Theorem 1 has another fascinating consequence. Assume that we have a graph in which all nodes have an odd degree. If $\varepsilon < 1/\Delta$, we get a directed degree splitting in which each node v has outdegree either $\lfloor d(v)/2\rfloor$ or $\lceil d(v)/2\rceil$. Note that the number of nodes for which the outdegree is $\lfloor d(v)/2\rfloor$ has to be exactly $n/2$. We therefore get an efficient distributed algorithm to exactly divide the number of nodes into two parts of equal size in any odd-degree graph. For bounded-degree graphs, the algorithm even runs in time $O(\log n)$.

Our Method in a Nutshell. The main technical contribution is a distributed algorithm that partitions the edge set of a given graph in *edge-disjoint short paths* such that each node is the start or end of at most δ paths. We call such a partition a *path decomposition* and δ its *degree*. Now if we orient each path of a path decomposition with degree δ consistently, we obtain an orientation of discrepancy at most δ. Moreover, such an orientation can be computed in time which is linear in the maximum path length.

To study path decompositions in graph G, it is helpful to consider an auxiliary graph H in which each edge $\{u, v\}$ represents a path from u to v in G; now δ is the maximum degree of graph H. To construct a low-degree path decomposition where δ is small, we can start with a trivial decomposition $H = G$, and then repeatedly join pairs of paths: we can replace the edges $\{u, v_1\}$ and $\{u, v_2\}$ in graph H with an edge $\{v_1, v_2\}$, and hence make the degree of u lower, at a cost of increasing the path lengths—this operation is called a *contraction* here.

If each node u simply picked arbitrarily some edges $\{u, v_1\}$ and $\{u, v_2\}$ to contract, this might result in long paths or cycles. The key idea is that we can use a *high-outdegree orientation* to select a good set of edges to contract: Assume that we have an orientation in H such that all nodes have outdegree at least $2k$. Then each node could select k pairs of outgoing edges to contract; this would reduce the maximum degree of H from δ to $\delta - 2k$ and only double the maximum length of a path.

In essence, this idea makes it possible to *amplify* the quality of an orientation algorithm: Given an algorithm A that finds an orientation with a large (but not optimal) outdegree, we can apply A repeatedly to reduce the maximum degree of H. This will result in a low-degree path decomposition of G, and hence also provide us with a well-balanced orientation in G.

Outline. In Section 2 we show how to partitioning graphs into edge-disjoint short paths. In Section 3 we use these results to prove our main result on the distributed computation of edge-splittings (Theorem 1). The proof of Corollary 2 is deferred to the full version of the paper [10].

2 Short Path Decompositions

The basic building block of our approach is to find consistently oriented and short (length $O(\Delta)$) paths in an oriented graph. The first crucial observation is that an oriented path going through a node v is "good" from the perspective of v in the sense that it provides exactly one incoming and one outgoing edge to v. Another important feature is that flipping a consistently oriented path does not increase the discrepancy between incoming and outgoing edges for any non-endpoint node along the path. Following these observations, we recursively decompose a graph into a set of short paths, and merge the paths to ensure that every node is at the end of only a few paths. If a node v is at the end of $\delta(v)$ paths an arbitrary orientation of these paths will provide a split with discrepancy at most $\delta(v)$ for v.

The recursive graph operations may turn graphs into multigraphs with self-loops. Thus throughout the paper a multigraph is allowed to have self-loops and the nodes of a path $v_1, \ldots, v_k$ do not need to be distinct; however, a path can contain each edge at most once. A self-loop at a node v contributes two to the degree of v.

2.1 Orientations and Edge Contractions

The core concept to merge many paths in parallel in one step of the aforementioned recursion is given by the concept of weak $k(v)$-orientations. We begin by extending and adapting prior work[11] on weak orientations to our needs.

▶ **Definition 3.** A *weak $k(v)$-orientation* of a multigraph $G = (V, E)$ is an orientation of the edges E such that each node $v \in V$ has outdegree at least $k(v)$.

Note that a weak 1-orientation is a *sinkless orientation*. By earlier work, it is known that a weak 1-orientation can be found in time $O(\log n)$ in simple graphs of minimum degree at least three.

▶ **Lemma 4** (sinkless orientation, [11]). *A weak 1-orientation can be computed by a deterministic algorithm in $O(\log n)$ rounds in simple graphs with minimum degree 3 (and by a randomised algorithm in $O(\log \log n)$ rounds in the same setting).*

In our proofs, we may face multigraphs with multiple self-loops and with nodes of degree less than three and thus, we need a slightly modified version of this result.

▶ **Corollary 5** (sinkless orientation, [11]). *Let $G = (V, E)$ be a multigraph and $W \subseteq V$ a subset of nodes with degree at least three. Then, there is a deterministic algorithm that finds an orientation of the edges such that every node in W has outdegree of at least one and runs in $O(\log n)$ rounds (and a randomised algorithm that runs in $O(\log \log n)$ rounds).*

Proof. For every multi-edge, both endpoints pick one edge and orient it outwards, ties broken arbitrarily. For every self-loop, the node will orient it arbitrarily. This way, every node with an incident multi-edge or self-loop will have an outgoing edge.

From here on, let us ignore the multi-edges and self-loops and focus on the simple graph H remaining after removing the multi-edges. For every node v with degree at most two in H, we connect v to $3 - d(v)$ copies of the following gadget U. The set of nodes of $U = \{u_1, u_2, u_3, u_4, u_5\}$ is connected as a cycle. Furthermore, we add edges $\{u_2, u_4\}$ and $\{u_3, u_5\}$ to the gadget and connect u_1 to v. This way, the gadget is 3-regular.

In the simple graph constructed by adding these gadgets, we run the algorithm of Lemma 4. Thus, any node of degree at least three in the original graph that was not initially adjacent to a multi-edge or self-loop gets an outgoing edge. Since we know that every node incident to a multi-edge or self-loop in G also has an outgoing edge, the claim follows. ◀

The sinkless orientation algorithm from Corollary 5 immediately leads to an algorithm which finds a weak $\lfloor d(v)/3 \rfloor$-orientation in multigraphs in time $O(\log n)$.

▶ **Lemma 6** (weak $\lfloor d(v)/3 \rfloor$-orientation)**.** *There is a deterministic algorithm that finds a weak $\lfloor d(v)/3 \rfloor$-orientation in time $O(\log n)$ in multigraphs.*

Proof. Partition node v into $\lceil d(v)/3 \rceil$ nodes and split its adjacent edges among them such that $\lfloor d(v)/3 \rfloor$ nodes have exactly three adjacent edges each and the remaining node, if any, has $d(v) \bmod 3$ adjacent edges. Note that the partitioning may cause self-loops to go between two different copies of the same node. Then, use the algorithm from Corollary 5 to compute a weak 1-orientation of the resulting multigraph where degree two or degree one nodes do not have any outdegree requirements. If we undo the partition but keep the orientation of the edges we have a weak $\lfloor d(v)/3 \rfloor$-orientation of the original multigraph. ◀

The techniques in this section need orientations in which nodes have at least two outgoing edges. Lemma 6 provides such orientations for nodes of degree at least six; but for nodes of smaller degree it guarantees only one outgoing edge. It is impossible to improve this for nodes with degree smaller than five (cf. [10, Theorem 7.1]) in time $o(n)$. But we obtain the following result for the nodes with degree five. Its proof relies on different techniques than the techniques in this section, and therefore it is deferred to the full version of the paper.

▶ **Lemma 7** (outdegree 2)**.** *The following problem can be solved in time $O(\log n)$ with deterministic algorithms and $O(\log \log n)$ with randomised algorithms: given any multigraph, find an orientation such that all nodes of degree at least 5 have outdegree at least 2.*

The concept of weak orientations can be extended to both indegrees and outdegrees.

▶ **Definition 8.** A *strong $k(v)$-orientation* of a multigraph $G = (V, E)$ is an orientation of the edges E such that each node $v \in V$ has both indegree and outdegree at least $k(v)$.

2.2 Path Decompositions

We now introduce the concept of a path decomposition. The decomposition proves to be a strong tool due to the fact that it can be turned into a strong orientation (cf. Lemma 11).

▶ **Definition 9.** Given a multigraph $G = (V, E)$, a positive integer λ, and a function $\delta \colon V \to \mathbb{R}_{\geq 0}$, we call a partition $\mathcal{P}$ of the edges E into disjoint paths $P_1, \ldots, P_\rho$ a *(δ, λ)-path decomposition* if

- for every $v \in V$ there are at most $\delta(v)$ paths that start or end in v,
- each path P_i is of length at most λ.

For each path decomposition $\mathcal{P}$, we define the multigraph $G(\mathcal{P})$ as follows: the vertex set of $G(\mathcal{P})$ is V, and there is an edge between two nodes $u, v \in V$ if $\mathcal{P}$ has a path which starts at u and ends at v or vice versa. The *degree of v in $\mathcal{P}$* is defined to be its degree in $G(\mathcal{P})$ and the *maximum degree of the path decomposition* $\mathcal{P}$ is the maximum degree of $G(\mathcal{P})$.

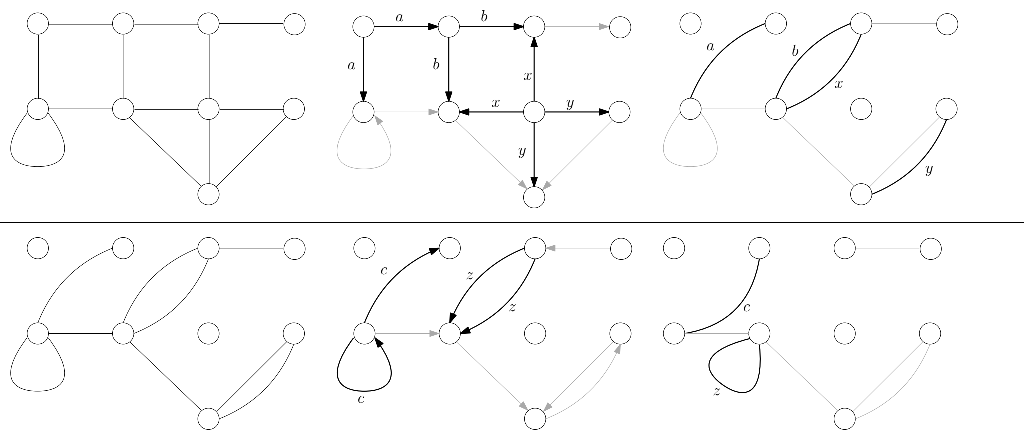

Figure 1 In two sequences of three illustrations this figure depicts two sets of contractions. In each line the first illustration is the situation before the contraction, the second one depicts the orientation and the selected outgoing edges which will be contracted in parallel and the third illustration shows the situation after the contraction where new edges are highlighted.
A contraction may produce isolated nodes, multi-edges and self loops. If a self loop $\{v, v\}$ is selected to be contracted with any other edge $\{v, w\}$ it simply results in a new edge $\{v, w\}$ as if the self loop was any other edge. Such a contraction still reduces the degree of v by two.
Note that we used a graph with small node degrees for illustration purposes. We cannot quickly compute an orientation with large outdegree for nodes with degree less than five.

Notice that $\delta(v)$ is an upper bound on the degree of v in $\mathcal{P}$ and $\max_{v\in V} \delta(v)$ is an upper bound on the maximum degree of the path decomposition. Note that $d_G(v) - d_{G(\mathcal{P})}(v)$ is always even. To make proofs more to the point instead of getting lost in notation, we often identify $G(\mathcal{P})$ with $\mathcal{P}$ and vice versa. A distributed algorithm has computed a path decomposition $\mathcal{P}$ if every node knows the paths of $\mathcal{P}$ it belongs to. Note that it is trivial to compute a $(d(v), 1)$-path decomposition in 1 round, because every edge can form a separate path.

Let $\lfloor \cdot \rfloor_*$ denote the function which rounds down to the previous even integer, that is, $\lfloor x \rfloor_* = 2\lfloor x/2 \rfloor$. The following virtual graph transformation, which we call *edge contraction*, is the core technical construction in this section.

Disjoint Edge Contraction

The basic idea behind edge contraction is to turn two incident edges $\{v, u\}$ and $\{v, w\}$ into a single edge $\{u, w\}$ by removing the edges $\{v, u\}$ and $\{v, w\}$ and adding a new edge $\{u, w\}$. We say that node v contracts when an edge contraction is performed on some pair of edges $\{v, u\}$ and $\{v, w\}$. When node v performs a contraction of edges $\{v, u\}$ and $\{v, w\}$, its degree $d(v)$ is reduced by two while maintaining the degrees of u and w. Furthermore, any set of nodes contracting any set of their incident edges at most doubles the distance between any pair of non-isolated nodes. Notice that adjacent nodes can only contract disjoint pairs of edges in parallel and a contraction may also produce isolated nodes, multi-edges and self-loops. If a self-loop $\{v, v\}$ is selected to be contracted with any other edge $\{v, w\}$ it simply results in a new edge $\{v, w\}$ as if the self-loop was any other edge. Such a contraction still reduces the degree of v by two as the self-loop was considered as both – an incoming and an outgoing edge of v. See Figure 1 for an illustration.

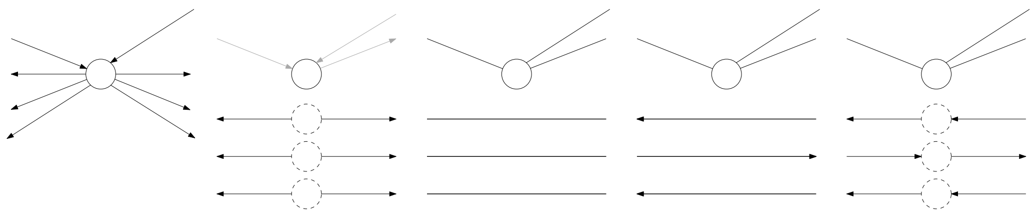

Figure 2 The first two illustrations show that selecting the outgoing edges for a contraction can be seen as dividing the node into a set of virtual nodes, each incident to two outgoing edges. Then, in the third illustration, the contraction is obtained by removing the virtual nodes but keeping the connection alive. The last two illustrations show how an orientation on contracted edges is used to orient the edges of the original graph such that virtual nodes obtain an equal split (and such that the original node obtains a good split).

Edge contractions can be used to compute path decompositions, e.g., an edge which is created through a contraction of two edges can be seen as a path of length two. If an edge $\{u, v\}$ represents a path from u to v in G, e.g., when recursively applying edge contractions on the graph $G(\mathcal{P})$ for some given path decomposition $\mathcal{P}$, each contraction merges two paths of $\mathcal{P}$. If each node simply picked arbitrarily some edges to contract, this might result in long paths or cycles. The key idea is to use orientations of the edges to find large sets of edges which can be contracted in parallel. If every node only contracts outgoing edges of a given orientation all contractions of all nodes can be performed in parallel.

If we start with a trivial decomposition, i.e., each edge is its own path, and perform k iterations of parallel contraction, where, in each iteration, each node contracts two edges, we obtain a $(d(v)-2k, 2^k)$-path decomposition. If we want the degrees $d(v)-2k$ to be constant we have to choose k, i.e., the number of iterations, in the order of Δ which implies exponentially long paths and runtime as the path lengths (might) double with each contraction.

The technical challenge to avoid exponential runtime is to achieve a lot of parallelism while at the same time reducing the degrees quickly. We achieve this with the help of weak orientation algorithms: An outdegree of $f(v)$ at node v allows the node to contract $\lfloor f(v) \rfloor_*$ edges at the same time and in parallel with all other nodes. If $f(v)$ is a constant fraction of $d(v)$ this implies that $O(\log \Delta)$ iterations are sufficient to reach a small degree. As the runtime is exponential in the number of iterations and the constant in the O-notation might be large, this is still not enough to ensure a runtime which is linear in Δ, up to polylogarithmic terms. Instead, we begin with the weak orientation algorithm from the previous section and iterate it until a path decomposition with a small (but not optimal!) degree is obtained. Then we use it to construct a better orientation algorithm. Then, we use this better orientation to compute an even better one and so on. Recursing with the correct choice of parameters leads to a runtime which is linear in Δ, up to polylogarithmic terms. We take the liberty to use the terms recursion and iteration interchangeably depending on which term is more suitable in the respective context. Refer to Figure 2 for an illustration of the edge contraction technique with a given orientation.

We will now apply a simple version of our contraction technique to obtain a fast and precise path decomposition algorithm in Δ-regular graphs for $\Delta = O(1)$. The result can also be formulated for non-regular graphs, but here we choose regular graphs to focus on the proof idea which is the key theme throughout most proofs of this section.

▶ **Theorem 10** ($(\Delta - 2k, 2^k)$-path decomposition). *Let $G = (V, E)$ be a Δ-regular multigraph. For any positive integer $k \leq \Delta/2 - 2$ there is a deterministic distributed algorithm that computes a $(\Delta - 2k, 2^k)$-path decomposition in time $O(2^k \log n)$.*

Proof. We recursively compute k multigraphs $H_1, \dots, H_k$ where H_k corresponds to the resulting path decomposition. To obtain H_1, we begin by computing a weak 2-orientation π of G with the algorithm from Lemma 6 (note that by assumption we have $k \geq 1$ and therefore $\Delta \geq 6$). Then, every node contracts a pair of outgoing incident edges. Notice that contractions of adjacent nodes are always disjoint. The degree of v is reduced to $\Delta - 2$ and each edge in the resulting multigraph H_1 consists of a path in G of length at most two.

Applying this method recursively with recursion depth k yields multigraphs $H_1, \dots, H_k$ where the maximum degree of H_i is $\Delta - 2i$ and each edge in H_i corresponds to a path in G of length at most 2^i. Thus, H_k corresponds to a $(\Delta - 2k, 2^k)$-path decomposition. Note that there is one execution of Lemma 6 in each recursion level and it provides a weak 2-orientation of the respective graph because the degree of each node is at least six due to $i \leq k \leq \Delta/2 - 2$.

One communication round in recursion level i can be simulated in 2^i rounds in the original graph. Thus, the runtime is dominated by the application of Lemma 6 in recursion level k which yields a time complexity of $O(2^k \log n)$. ◀

Next, we show how to turn a (δ, λ)-path decomposition efficiently into a strong orientation. The strong orientation obtained this way has $\delta(v)$ as an upper bound on the discrepancy between in- and outdegree of node v.

▶ **Lemma 11.** *Let $G = (V, E)$ be a multigraph with a given (δ, λ)-path decomposition $\mathcal{P}$. There is a deterministic algorithm that computes a strong $\frac{1}{2}(d(v) - \delta(v))$-orientation of G in $O(\lambda)$ rounds.*

Proof. Let $H = G(\mathcal{P})$ be the virtual graph that corresponds to $\mathcal{P}$ and let π_H be an arbitrary orientation of the edges of H. Let (u, v) be an edge of H oriented according to π_H and let $P = u_1, \dots, u_k$, where $u_1 = u$ and $u_k = v$, be the path in the original graph G that corresponds to edge (u, v) in H. Now, we orient the path P in a consistent way according to the orientation of (u, v), i.e., edge $\{u_i, u_{i+1}\}$ is directed from u_i to u_{i+1} for all $1 \leq i \leq k$. Since every edge in G belongs to exactly one path in the decomposition, performing this operation for every edge in H provides a unique orientation for every edge in G. Let us denote the orientation obtained this way by π_G.

Consider some node v and observe that orienting any path that contains v but where v is not either the start or the endpoint adds exactly one incoming edge and one outgoing edge for v. Therefore, the discrepancy of the indegrees and outdegrees of v in π_G is bounded from above by the discrepancy in π_H, which is at most $\delta(v)$ by the definition of a (δ, λ)-path decomposition. It follows that π_G is a strong $\frac{1}{2}(d(v) - \delta(v))$-orientation.

Finally, since the length of any path in $\mathcal{P}$ is bounded above by λ, consistently orienting the paths takes λ communication rounds finishing the proof. ◀

In the following, we formally use weak orientations to compute a path decomposition. This lemma will later be iterated in Corollary 13

▶ **Lemma 12.** *Assume that there exists a deterministic distributed algorithm that finds a weak $\left(\left(\frac{1}{2} - \varepsilon\right)d(v) - 2\right)$-orientation in time $T(n, \Delta)$.*

Then, there is a deterministic distributed algorithm that finds a $\left(\left(\frac{1}{2} + \varepsilon\right)d(v) + 4,\, 2\right)$-path decomposition in time $O(T(n, \Delta))$.

Proof. Let G be a multigraph with a weak $\left(\left(\frac{1}{2} - \varepsilon\right)d(v) - 2\right)$-orientation given by the algorithm promised in the lemma statement. Now every node v arbitrarily divides the outgoing edges into pairs and contracts these pairs yielding a multigraph with degree at most

$$d(v) - \left\lfloor \left(\tfrac{1}{2} - \varepsilon\right)d(v) \right\rfloor_* + 2 \leq \left(\tfrac{1}{2} + \varepsilon\right)d(v) + 4.$$

Observing that all of the chosen edge pairs are disjoint yields that the constructed multigraph is a $\big((\frac{1}{2}+\varepsilon)d(v)+4,\, 2\big)$-path decomposition. The contraction operation requires one round of communication. ◀

In the following corollary we iterate Lemma 12 to obtain an even better path decomposition. Furthermore, more care is required in the details to avoid rounding errors and to obtain the correct result when the degrees get small. Corollary 13 will be applied many times in proceeding subsections.

▶ **Corollary 13.** *Let $0 < \varepsilon \leq 1/6$. Assume that $T(n,\Delta) \geq \log n$ is the running time of an algorithm $\mathcal{A}$ that finds a weak $\big((1/2-\varepsilon)d(v)-2\big)$-orientation. Then for any positive integer i, there is a deterministic distributed algorithm $\mathcal{B}$ that finds a $\big((1/2+\varepsilon)^i d(v)+4,\, 2^{i+5}\big)$-path decomposition $\mathcal{P}$ in time $O(2^i \cdot T(n,\Delta))$.*

Proof. Let i be a positive integer. We define algorithm $\mathcal{B}$ such that it uses algorithm $\mathcal{A}$ to recursively compute graphs $H_0, H_1, \ldots, H_i, H_{i+1}, \ldots, H_{i+5}$ and path decompositions $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_i, \mathcal{P}_{i+1}, \ldots, \mathcal{P}_{i+5}$. Let $G = (V, E)$ be a multigraph. For $j = 0, \ldots, i-1$ we set $H_0 = G$ and $H_{j+1} = H_j(\mathcal{P}_{j+1})$, where $\mathcal{P}_{j+1}$ is the path decomposition which is returned by applying Lemma 12 with algorithm $\mathcal{A}$ on H_j. This guarantees that path decomposition $\mathcal{P}_i$ has maximum degree $(\frac{1}{2}+\varepsilon)^i d(v)+12$. The remaining five graph decompositions are computed afterward (see the end of this proof) and reduce the additive 12 to an additive 4.

Properties of $\mathcal{P}_1, \ldots, \mathcal{P}_i$. We first show that for $j = 1, \ldots, i$ the path decomposition $\mathcal{P}_j$ is a $(z_j(v), 2^j)$-path decomposition with

$$z_j(v) = \big(\tfrac{1}{2}+\varepsilon\big)^j d(v) + 4\sum_{k=0}^{j-1}\big(\tfrac{1}{2}+\varepsilon\big)^k.$$

With every application of Lemma 12 the length of the paths at most double in length which implies that the path length of $\mathcal{P}_j$ is upper bounded by 2^j. We now prove by induction that the variables $z_j(v)$, $j = 1, \ldots i$ behave as claimed:

- *Base case:* $z_1(v) = \big(\frac{1}{2}+\varepsilon\big)d(v)+4$ follows from the invocation of Lemma 12 with $\mathcal{A}$ on $H_0 = G$.
- *Inductive step:* Using the properties of Lemma 12 we obtain

$$\begin{aligned} z_{j+1}(v) = \big(\tfrac{1}{2}+\varepsilon\big)z_j(v)+4 &\leq \big(\tfrac{1}{2}+\varepsilon\big)\left(\big(\tfrac{1}{2}+\varepsilon\big)^j d(v) + 4\sum_{k=0}^{j-1}\big(\tfrac{1}{2}+\varepsilon\big)^k\right)+4 \\ &= \big(\tfrac{1}{2}+\varepsilon\big)^{j+1} d(v) + 4\sum_{k=0}^{j}\big(\tfrac{1}{2}+\varepsilon\big)^k. \end{aligned}$$

Using the geometric series to bound the last sum and then $\varepsilon \leq 1/6$ we obtain that

$$z_i(v) \leq \big(\tfrac{1}{2}+\varepsilon\big)^i d(v) + 12.$$

Reducing the Additive Term. Now, we compute the five further path decompositions $\mathcal{P}_{i+1}, \ldots, \mathcal{P}_{i+5}$ to reduce the additive term in the degrees of the path decomposition from 12 to 4; in each path decomposition this additive term is reduced by two for certain nodes. In each of the first four path decompositions nodes with degree at least six in the current path decomposition reduce the additive term by at least two: we compute a weak $\lfloor d(v)/3 \rfloor$-orientation (using Lemma 6) and then every node with degree at least six contracts two outgoing edges. In the last path decomposition we compute an orientation in which every node with degree at least five in the current path decomposition has two outgoing edges (using Lemma 7) and then each of them contracts two incident edges. Thus in the last path decomposition the additive term of nodes with degree five is reduced by two.

To formally prove that we obtain the desired path decomposition let $x_{i+j}(v)$ be the actual degree of node v in $G(\mathcal{P}_{i+j})$ for $j = 0, \ldots, 5$. First note that the degree of a node never increases due to an edge contraction, not even due to an edge contraction which is performed by another node.

Constructing $\mathcal{P}_{i+1}, \ldots, \mathcal{P}_{i+4}$. To compute path decomposition $\mathcal{P}_{i+j+1}$, $j = 0, \ldots, 3$, we compute an orientation of $G(\mathcal{P}_{i+j})$ in which every node v with $x_{i+j}(v) \geq 6$ has outdegree at least two (one can use the algorithm described in Lemma 6). Then $\mathcal{P}_{i+j+1}$ is obtained if every node with $x_{i+j}(v) \geq 6$ contracts two of its incident outgoing edges. So, whenever $x_{i+j}(v) \geq 6$ we obtain that $x_{i+j+1}(v) = x_{i+j}(v) - 2$, that is $x_{i+j+1} \leq z_i(v) - 2(j+1)$. If $x_{i+j}(v) \geq 6$ for all $j = 0, \ldots, 3$ we have

$$x_{i+5}(v) \leq x_{i+4}(v) \leq (1/2 + \varepsilon)^i d(v) + 4.$$

Otherwise, for some $j = 0, \ldots, 3$, we have $x_{i+j}(v) \leq 5$, that is, $x_{i+4}(v) \leq 4$ or $x_{i+4}(v) = 5$. If $x_{i+4}(v) \leq 4$ we have

$$x_{i+5}(v) \leq x_{i+4}(v) \leq 4 \leq (1/2 + \varepsilon)^i d(v) + 4.$$

Constructing $\mathcal{P}_{i+5}$. For nodes with $x_{i+4}(v) = 5$ we compute one more path decomposition. We use Lemma 7 to compute an orientation of $G(\mathcal{P}_4)$ in which each node with degree at least five has two outgoing edges; then each node with at least two outgoing edges contracts one pair of its incident outgoing edges. Thus the degree of nodes with degree five reduces by two and we obtain that the path decomposition $\mathcal{P}_{i+5}$ is a $\big((\frac{1}{2} + \varepsilon)^i d(v) + 4,\, 2^{i+5}\big)$-path decomposition.

Running Time. The time complexity to invoke algorithm $\mathcal{A}$ or the algorithms from Lemma 6 or Lemma 7 on graph H_j is $O(2^j T(n, \Delta))$ because the longest path in H_j has length 2^j and $T(n, \Delta) \geq \log n$. Thus, the total runtime is

$$O\Bigg(\sum_{j=0}^{i+5} 2^j T(n, \Delta)\Bigg) = O\big(2^i T(n, \Delta)\big). \qquad \blacktriangleleft$$

2.3 Amplifying Weak Orientation Algorithms

Now, we use Corollary 13 to iterate a given weak orientation algorithm $\mathcal{A}$ to obtain a new weak orientation algorithm $\mathcal{B}$. The goal is that $\mathcal{B}$ has an outdegree guarantee which is much closer to $(1/2)d(v)$ than the guarantee provided by algorithm $\mathcal{A}$.

Let $0 < \varepsilon_2 < \varepsilon_1 \leq \frac{1}{6}$, $\alpha = \frac{1}{2} - \varepsilon_1$, and $\beta = \frac{1}{2} + \varepsilon_1$. The roadmap for the proofs of this section is as follows:

- In the proof of Lemma 15:
  1. Execute i iterations of a weak $(\alpha d(v) - 2)$-orientation algorithm, for an i that will be chosen later, and after each iteration, perform disjoint edge contractions. Thus, obtain a $\big(\beta^i d(v) + 4,\, 2^{i+5}\big)$-path decomposition using Corollary 13.
  2. Apply Lemma 11 to obtain a weak $\big(\frac{1}{2}(1 - \beta^i)d(v) - 2\big)$-orientation.

3. By setting $i = \log(\varepsilon_2)/\log(\beta)$ we get that $\beta^i = \varepsilon_2$ and the running time of steps 1–2 is

$$O(2^i T(n,\Delta)) = O\big(\varepsilon_2^{\log_2^{-1}\beta} \cdot T(n,\Delta)\big) = O\big(\varepsilon_2^{-(1+24\varepsilon_1)} \cdot T(n,\Delta)\big),$$

where $T(n,\Delta)$ is the runtime of the weak $(\alpha d(v)-2)$-orientation algorithm. The last equality holds because with Lemma 14, we obtain that $-\log_2^{-1}\beta \le 1+24\varepsilon_1$ when $\varepsilon_1 \le 1/6$.

- In the proof of Theorem 16:

4. Use Lemma 11 with $\varepsilon_1 = 1/6$ and $\varepsilon_2 = 1/\log\log\Delta$ to obtain an algorithm which computes a weak $\big((\frac{1}{2}-1/\log\log\Delta)d(v)-2\big)$-orientation and runs in time $O((\log\log\Delta)^{1.71}\cdot\log n)$. In this step, we plug in $\varepsilon_1 = 1/6$ to obtain the exponent

$$-\log_2^{-1}\beta = -\log_2^{-1}\big(\tfrac{1}{2}+\tfrac{1}{6}\big) < 1.71.$$

5. Using the construction twice more, once with $\varepsilon_1 = 1/\log\log\Delta$ and $\varepsilon_2 = 1/\log\Delta$ and once with $\varepsilon_1 = 1/\log\Delta$ and $\varepsilon_2 = 1/\Delta$, yields a weak $\big(\big(\frac{1}{2}-\frac{1}{\Delta}\big)\,d(v)-2\big)$-orientation algorithm that runs in time $O(\Delta\cdot\log\Delta\cdot(\log\log\Delta)^{1.71}\cdot\log n)$.

The following technical result is used to simplify running times; it is proved in the full version of the paper with a Taylor expansion.

▶ **Lemma 14.** *Let $0 < \varepsilon \le 1/6$. Then, $-\log_2^{-1}\left(\frac{1}{2}+\varepsilon\right) \le 1+24\varepsilon$.*

In the following lemma we perform steps 1–3 of the aforementioned agenda.

▶ **Lemma 15.** *Let $0 < \varepsilon_2 < \varepsilon_1 \le \frac{1}{6}$. Assume that there is a deterministic algorithm $\mathcal{A}$ which computes a weak $\big(\big(\frac{1}{2}-\varepsilon_1\big)\,d(v)-2\big)$-orientation and runs in time $T(n,\Delta)$. Then there is a deterministic weak $\big(\big(\frac{1}{2}-\varepsilon_2\big)\,d(v)-2\big)$-orientation algorithm $\mathcal{B}$ with running time*

$$O\Big(\varepsilon_2^{\log_2^{-1}(\frac{1}{2}+\varepsilon_1)}\cdot T(n,\Delta)\Big) = O\Big(\varepsilon_2^{-(1+24\varepsilon_1)}\cdot T(n,\Delta)\Big). \tag{1}$$

Proof. Let $i = \log_2(\varepsilon_2)/\log_2(1/2+\varepsilon_1)$. By Lemma 14, we get that $i \le (1+24\varepsilon_1)\log_2(1/\varepsilon_2)$; thus it is sufficient to show the left hand side of (1). By applying Corollary 13 with parameter i and algorithm $\mathcal{A}$, we get a distributed algorithm that finds a $\big((1/2+\varepsilon_1)^r\,d(v)+4,\;2^{i+5}\big)$-path decomposition in time

$$O\big(2^i\cdot T(n,\Delta)\big) = O\Big(\varepsilon_2^{\log_2^{-1}(\frac{1}{2}+\varepsilon_1)}\cdot T(n,\Delta)\Big).$$

The degree of node v in the path decomposition is upper bounded by $\big(\frac{1}{2}+\varepsilon_1\big)^i d(v)+4 = \varepsilon_2 d(v)+4$. Now Lemma 11 yields a weak $\big(\frac{1}{2}\big(1-\varepsilon_2\big)d(v)-2\big)$-orientation algorithm with the same running time; in particular, this is a weak $\big(\big(\frac{1}{2}-\varepsilon_2\big)d(v)-2\big)$-orientation algorithm. ◀

We close the section by performing steps 4–5 of the agenda. Note that the theorem is more general than the outlined agenda as it contains an additional parameter δ which can be used to tune the running time at the cost of the quality of the weak orientation algorithm.

▶ **Theorem 16.** *Let δ be a positive integer. There exist the following deterministic weak orientation algorithms.*

(a) *$\mathcal{A}$: weak $\big(\big(\frac{1}{2}-1/\log\log\frac{\Delta}{\delta}\big)\,d(v)-2\big)$-orientation in time $O\big(\big(\log\log\frac{\Delta}{\delta}\big)^{1.71}\cdot\log n\big)$.*

(b) *$\mathcal{B}$: weak $\big(\big(\frac{1}{2}-1/\log\frac{\Delta}{\delta}\big)\,d(v)-2\big)$-orientation in time $O\big(\log\frac{\Delta}{\delta}\cdot\big(\log\log\frac{\Delta}{\delta}\big)^{1.71}\cdot\log n\big)$.*

(c) *$\mathcal{C}$: weak $\big(\big(\frac{1}{2}-\frac{\delta}{\Delta}\big)\,d(v)-2\big)$-orientation in time $O\big(\frac{\Delta}{\delta}\cdot\log\frac{\Delta}{\delta}\cdot\big(\log\log\frac{\Delta}{\delta}\big)^{1.71}\cdot\log n\big)$.*

Proof. Each statement is proven by applying Lemma 15 with different values for ε_1 and ε_2.

(a) We obtain the algorithm $\mathcal{A}$ by applying Lemma 15 with the weak $\lfloor \Delta/3 \rfloor$-orientation algorithm from Lemma 6, that is with $\varepsilon_1 = 1/6$, and with $\varepsilon_2 = 1/\log\log(\Delta/\delta)$.
(b) Algorithm $\mathcal{B}$ is obtained by applying Lemma 15 with $\mathcal{A}$, that is $\varepsilon_1 = 1/\log\log(\Delta/\delta)$ and $\varepsilon_2 = 1/\log(\Delta/\delta)$.
(c) Algorithm $\mathcal{C}$ is obtained by applying Lemma 15 with $\mathcal{B}$, that is $\varepsilon_1 = 1/\log(\Delta/\delta)$ and $\varepsilon_2 = 1/(\Delta/\delta) = \delta/\Delta$. ◀

2.4 Short and Low Degree Path Compositions Fast

Our higher level goal is to compute a path decomposition where the degree is as small as possible to obtain a directed split with the discrepancy as small as possible (with methods similar to Lemma 11, also see the proof of Theorem 1). As we will show in the next theorem, with the methods introduced in this section and the appropriate choice of parameters, we can push the maximum degree of the path decomposition down to $\varepsilon d(v) + 4$ for any $\varepsilon > 0$. This is the true limit of this approach because we cannot compute weak 2-orientations of 4-regular graphs in sublinear time (see [10, Theorem 7.1]).

▶ **Theorem 17.** *Let $G = (V, E)$ be a multigraph with maximum degree Δ. For any $\varepsilon > 0$ there is a deterministic distributed algorithm which computes a $(\delta(v), O(1/\varepsilon))$-path decomposition in time $O\big(\alpha \cdot \log \alpha \cdot (\log\log \alpha)^{1.71} \cdot \log n\big)$, where $\alpha = 2/\varepsilon$ and $\delta(v) = \varepsilon d(v) + 3$ if $\varepsilon d(v) \geq 1$ and $\delta(v) = 4$ otherwise.*

Proof. Apply Corollary 13 with algorithm $\mathcal{B}$ from Corollary 16, $\delta = \Delta/\alpha$, and

$$i = \frac{\log \alpha^{-1}}{\log(1/2 + 1/\log(\alpha))}.$$

This implies a path decomposition with degrees $\lfloor \alpha^{-1} d(v) + 4 \rfloor = \lfloor \varepsilon d(v)/2 + 4 \rfloor$. If $\varepsilon d(v) \geq 1$ this is smaller than $\varepsilon d(v) + 3$. If $\varepsilon d(v) < 1$ this is at most 4. The length of the longest path is upper bounded by $O(2^i) = O\big(\alpha^{1+24/\log \alpha}\big) = O(\alpha)$ where we used Lemma 14. The runtime is bounded by $O\big(2^i \cdot T_{\mathcal{B}}(n, \Delta)\big) = O\big(\alpha \cdot \log \alpha \cdot (\log\log \alpha)^{1.71} \cdot \log n\big)$, where $T_{\mathcal{B}}(n, \Delta)$ is the running time of algorithm $\mathcal{B}$. ◀

Choosing $\varepsilon = 1/(2\Delta)$ in Lemma 17 yields the following corollary.

▶ **Corollary 18** (constant degree path decomposition). *There is a deterministic algorithm which computes a $(4, O(\Delta))$-path decomposition in time $O\big(\Delta \cdot \log \Delta \cdot (\log\log \Delta)^{1.71} \cdot \log n\big)$.*

▶ Remark. For any positive integer k smaller than $\log^*(\alpha) \pm O(1)$ one can improve the runtime of Lemma 17 to $O\big(\alpha \cdot (\log^{(k)} \alpha)^{0.71} \cdot \log n \cdot \Pi_{j=1}^{k} \log^{(j)} \alpha\big)$, where $\log^{(j)}(\cdot)$ denotes the j times iterated logarithm, $\alpha = 2/\epsilon$ and the constant in the O-notation grows exponentially in k. This essentially follows from a version of Theorem 16 that turns a weak $\big((1/2 - 1/\log^{(k)} \alpha) d(v) - 2\big)$-orientation algorithm into a weak $\big((1/2 - 1/\log \alpha) d(v) - 2\big)$-orientation algorithm in $k - 1$ iterations.

3 Directed and Undirected Splits

First note that an arbitrary consistent orientation of the paths in the best path decomposition of Section 2 would result in a splitting in which each node v has discrepancy at most $\varepsilon \cdot d(v) + 4$. In the case of directed splitting we slightly tune this by consistently orienting the paths in such a way that each node has at least one outgoing and one incoming path. As the graph corresponding to the path decomposition is a low degree graph this is the same as finding sinkless and sourceless orientations in low-degree graphs; the following corollary states that these orientations can be computed efficiently. Its proof can be found in the full version of the paper [10].

▶ **Corollary 19** (sinkless and sourceless orientation). *The following problem can be solved in time $O(\log n)$ with deterministic algorithms and $O(\log\log n)$ with randomised algorithms: given any multigraph, find an orientation such that all nodes of degree at least 3 have outdegree and indegree at least 1.*

We are now ready to prove our main result:

▶ **Theorem 1.** *For every $\varepsilon > 0$, there are deterministic $O\big(\varepsilon^{-1}\cdot\log\varepsilon^{-1}\cdot\big(\log\log\varepsilon^{-1}\big)^{1.71}\cdot\log n\big)$-round distributed algorithms for computing directed and undirected degree splittings with the following properties:*

(a) *For directed degree splitting, the discrepancy at each node v of degree $d(v)$ is at most $\varepsilon \cdot d(v) + 1$ if $d(v)$ is odd and at most $\varepsilon \cdot d(v) + 2$ if $d(v)$ is even.*

(b) *For undirected degree splitting, the discrepancy at each node v of degree $d(v)$ is at most $\varepsilon \cdot d(v) + 4$.*

Proof. For both parts apply Lemma 17, which provides a $\big(\delta(v), O(1/\varepsilon)\big)$-path decomposition $\mathcal{P}$ with $\delta(v) = \varepsilon d(v) + 3$ if $\varepsilon d(v) \geq 1$ and $\delta(v) = 4$ otherwise.

Proof of (b) Nodes color each path of $\mathcal{P}$ alternating with red and blue. Because the length of a path in $\mathcal{P}$ is bounded by $O(1/\varepsilon)$ this can be done in $O(1/\varepsilon)$ rounds.

Consider some node v and observe that v has one red and one blue edge for any path where v is not a startpoint or endpoint. Thus the discrepancy of node v is bounded above by $\delta(v) \leq \varepsilon d(v) + 4$.

Proof of (a) Use Corollary 19 to compute an orientation $\pi_{\mathcal{P}}$ of $G(\mathcal{P})$ in which all nodes which have degree at least three in $G(\mathcal{P})$ have at least one incoming and one outgoing edge. Then orient paths in the original graph according to $\pi_{\mathcal{P}}$ as in the proof of Lemma 11 and denote the resulting orientation of the edges of G with π_G.

Consider some node v and observe that orienting any path that contains v but where v is not a startpoint or endpoint adds exactly one incoming edge and one outgoing edge for v. Therefore, the discrepancy of the indegrees and outdegrees of v in $\pi_{\mathcal{P}}$ bounds from above the discrepancy of the indegrees and outdegrees in π_G. The goal is to upper bound this discrepancy as desired.

Therefor let $d_{\mathcal{P}}(v)$ denote the degree of v in $G(\mathcal{P})$. If $d_{\mathcal{P}}(v)$ is at least three then its discrepancy in $\pi_{\mathcal{P}}$ is bounded by $d_{\mathcal{P}}(v) - 2$ as the algorithm from Corollary 19 provided one incoming and one outgoing edge for v in $G(\mathcal{P})$. Furthermore we obtain that $d_{\mathcal{P}}(v)$ and $d(v)$ have the same parity because $d(v) = d_{\mathcal{P}}(v) + 2x$ holds where x is the number of paths that contain v but where v is neither a startpoint nor an endpoint. We have the following cases.

- $d_{\mathcal{P}}(v) \geq 3$:
  - $\varepsilon d(v) \geq 1$: v's discrepancy in π_G is bounded by $d_{\mathcal{P}}(v) - 2 \leq \varepsilon d(v) + 1$.
  - $\varepsilon d(v) < 1$, $d(v)$ even: v's discrepancy in π_G is bounded by $d_{\mathcal{P}}(v) - 2 \leq 2$.
  - $\varepsilon d(v) < 1$, $d(v)$ odd: As $d_{\mathcal{P}}(v)$ has to be odd and $3 \leq d_{\mathcal{P}}(v) \leq \delta(v) = 4$ holds we have $d_{\mathcal{P}}(v) = 3$. Thus v's discrepancy in π_G is bounded by $d_{\mathcal{P}}(v) - 2 \leq 1$.

- $d_{\mathcal{P}}(v) < 3$:
  - $d(v)$ even: We have $d_{\mathcal{P}} \in \{0, 2\}$ and v's discrepancy in π_G is also 0 or 2.
  - $d(v)$ odd: We have $d_{\mathcal{P}} = 1$ and v's discrepancy in π_G is also 1.

In all cases we have that the discrepancy of node v is upper bounded by $\varepsilon d(v) + 1$ if $d(v)$ is even and by $\varepsilon d(v) + 2$ if $d(v)$ is even, which proves the result. ◀

References

1 Leonid Barenboim and Michael Elkin. Distributed deterministic edge coloring using bounded neighborhood independence. In *Proc. PODC 2011*, pages 129–138, 2011. doi:10.1007/s00446-012-0167-7.

2 József Beck and Tibor Fiala. "Integer-making" theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981. doi:10.1016/0166-218X(81)90022-6.

3 Debe Bednarchak and Martin Helm. A note on the beck-fiala theorem. *Combinatorica*, 17(1):147–149, 1997.

4 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In *Proc. STOC 2016*, pages 479–488, 2016. doi:10.1145/2897518.2897570.

5 Boris Bukh. An improvement of the Beck–Fiala theorem. *Combinatorics, Probability & Computing*, 25(03):380–398, 2016. doi:10.1017/S0963548315000140.

6 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. In *Proc. FOCS 2016*, pages 615–624, 2016. doi:10.1109/FOCS.2016.72.

7 Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity.* Cambridge University Press, 2000.

8 Andrzej Czygrinow, Michał Hańćkowiak, and Michał Karoński. Distributed $O(\Delta \log n)$-edge-coloring algorithm. In *Proc. ESA 2001*, pages 345–355, 2001. doi:10.1007/3-540-44676-1_29.

9 Yefim Dinitz. Dinitz' algorithm: The original version and Even's version. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, pages 218–240. Springer, 2006. doi:10.1007/11685654_10.

10 M. Ghaffari, J. Hirvonen, F. Kuhn, Y. Maus, J. Suomela, and J. Uitto. Improved Distributed Degree Splitting and Edge Coloring. *ArXiv e-prints*, June 2017. arXiv:1706.04746.

11 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In *Proc. SODA 2017*, pages 2505–2523, 2017.

12 Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM J. Discrete Math.*, 15(1):41–57, 2001. doi:10.1137/S0895480100373121.

13 Amos Israeli and Yossi Shiloach. An improved parallel algorithm for maximal matching. *Inf. Process. Lett.*, 22(2):57–60, 1986. doi:10.1016/0020-0190(86)90141-9.

14 Howard J. Karloff and David B. Shmoys. Efficient parallel algorithms for edge coloring problems. *J. Algorithms*, 8(1):39–52, 1987. doi:10.1016/0196-6774(87)90026-5.

15 Nathan Linial. Distributive graph algorithms—global solutions from local data. In *Proc. FOCS 1987*, pages 331–335, 1987. doi:10.1109/SFCS.1987.20.

16 Moni Naor and Larry Stockmeyer. What can be computed locally? In *Proc. STOC 1993*, pages 184–193, 1993. doi:10.1145/167088.167149.

17 David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000. doi:10.1137/1.9780898719772.