Distributed Graph Algorithms

Computer Science, ETH Zurich

Mohsen Ghaffari

These are draft notes, used as supplementary material for the "Principles of Distributed Computing" course at ETH Zurich. The notes mainly present the technical content and are missing, in several places, the introductory explanations such as the underlying motivation and the learning goals (which are discussed in the class). The notes will be updated regularly. Feedback and comments would be greatly appreciated and should be emailed to ghaffari@inf.ethz.ch.

Last update: August 13, 2020

Contents

Chapter 1

Local Problems

1.1 Introduction & the LOCAL Model

In this chapter, we discuss distributed algorithms for some of the fundamental local graph problems, such as graph coloring, maximal independent set, maximal matching, and network decomposition. In an informal sense, by calling these local problems, we mean that we will be able to find algorithms for them where the output of each node (e.g., its color) will depend only on the toplogy of a small neighborhood around it in the network, rather than the entire network.

We work with the LOCAL model, which was first formalized by Linial [Lin87, Lin92]. The model definition is as follows.

Definition 1.1. *(The LOCAL model) We consider an arbitrary* n*-node graph* $G = (V, E)$ *where* $V = \{1, 2, \ldots, n\}$*, which abstracts the communication network. Unless noted otherwise,* G *is a simple, undirected, and unweighted graph. There is one process on each node* $v \in V$ *of the network. At the beginning, the processes do not know the graph* G*, except for knowing*[1] n*, and their own unique identifier in* $\{1, 2, \ldots, n\}$*. The algorithms work in synchronous rounds. Per round, each node/process performs some computation based on its own knowledge, and then sends a message to all of its neighbors, and then receives the messages sent to it by its neighbors in that round. In each graph problem in this model, we require that each node learns its own part of the output, e.g., its own color in a graph coloring.*

Comment: We stress that the model does not assume any limitation on the size of the messages, or on the computational power of the processes. Because of this, it is not hard to see that, any t-round algorithm in the LOCAL model induces a function which maps the t-hop neighborhood of each node to its output (why?). For instance, a t-round algorithm for graph coloring maps the topology induced by vertices within distance t of a vertex v to the coloring of vertex v. The converse of this statement is also true, meaning that if for a given graph problem, such a function exists, then there is also a t-round algorithm for solving that problem. Hence, one can say that the LOCAL model captures the *locality* of graph problems in a mathematical sense.

Observation 1.2. *Any graph problem on any* n*-node graph* G *can be solved in* $O(n)$ *rounds. In fact, using* D *to denote the diameter of the graph, any problem can be solved in* $O(D)$ *rounds.*

[1]Most often, the algorithms will use only the assumption that nodes know an upper bound N on n such that $N \in [n, n^c]$ for a small constant $c \geqslant 1$.

1.2 Coloring Rooted Trees

We start by examining the graph coloring problem, and in a very special case, coloring rooted trees. This basic-looking problem already turns out to have some quite interesting depth, as we see in this section.

The setting is as follows. We consider an arbitrary rooted tree $T = (V, E)$, such that $V = \{1, 2, \ldots, n\}$, and where each node v knows its parent $p(v)$ in T. The objective is to find a proper coloring of T, that is, a color assignment $\phi : V \to \{1, 2, \ldots, q\}$ such that there does not exist any node v with $\phi(v) = \phi(p(v))$. Of course, we are interested in using a small number of colors q, and we seek fast algorithms for computing such a coloring, that is, algorithms that use a small number of rounds.

Clearly, each tree can be colored using just 2 colors. However, computing a 2-coloring in the LOCAL model is not such an interesting problem, due to the following simple observation:

Observation 1.3. *Any* LOCAL *algorithm for 2-coloring an n-node directed path requires at least $\Omega(n)$ rounds.*

In contrast, 3-coloring has no such unfortunate lower bound, and in fact, entails something quite non-trivial: it has a tight round complexity of $\frac{1}{2}\log^* n \pm O(1)$. Recall the definition of the log-Star function:

$$\log^*(x) = \begin{cases} 0 & \text{if } x \leqslant 1 \\ 1 + \log^*(\log x) & \text{if } x > 1 \end{cases}$$

To prove this tight $\frac{1}{2}\log^* n \pm O(1)$ round complexity, in the next two subsections, we explain the following two directions of this result:

- First, in Section 1.2.1, we explain a $\log^* n + O(1)$ round algorithm for 3-coloring rooted trees. The upper bound can actually be improved to $\frac{1}{2}\log^* n + O(1)$ rounds [SV93], and even to exactly $\frac{1}{2}\log^* n$ rounds [RS14], but we do not cover those refinements. There are four known methods for obtaining $O(\log^* n)$-round algorithms [CV86, SV93, NS93, FHK16]. The algorithm we describe is based on an idea of [NS95] and some extra step from [GPS87]. The approach of [CV86] will be covered in Exercise 1.1.

- Then, in Section 1.2.2, we prove the above bound to be essentially optimal by showing that any deterministic algorithm for 3-coloring rooted trees requires at least $\frac{1}{2}\log^* n - O(1)$ rounds. This result was first proved by [Lin87, Lin92]. We explain a somewhat streamlined proof, based on [LS14]. The lower bound holds also for randomized algorithms [Nao91], but we will not cover that generalization, for the sake of simplicity. Furthermore, essentially the same lower bound can be obtained as a direct corollary of *Ramsey Theory*. We will have a brief explanation about that, at the end of this subsection.

1.2.1 3-Coloring Rooted Trees in $\log^* n + O(1)$ Rounds

Theorem 1.4. *Any n-node rooted-tree can be colored with 3 colors, in $\log^* n + O(1)$ rounds.*

Notice that the initial numbering $1, 2, \dots, n$ of the vertices is already a coloring with n colors. We explain a method for gradually improving this coloring, by iteratively reducing the number of colors. The key ingredient is a single-round color-reduction method, based on *Sperner families*, which achieves the following:

Lemma 1.5. *Given a k-coloring ϕ_{old} of a rooted tree where $k \geqslant C_0$ for a constant*[2] *C_0, in a single round, we can compute a k'-coloring ϕ_{new}, for $k' = \log k + \log\log k/2 + 1$.*

Proof. Let each node u send its color $\phi_{old}(u)$ to its children. We now describe a method which allows each node v to compute its new coloring $\phi_{new}(v)$, based on $\phi_{old}(v)$ and $\phi_{old}(u)$ where u is the parent of v, with no further communication.

Consider an arbitrary one-to-one mapping $M : \{1, 2, \dots, k\} \to \mathcal{F}_{k'}$, fixed a priori, where $\mathcal{F}_{k'}$ denotes the set of all the subset of size $k'/2$ of the set $\{1, 2, \dots, k'\}$. Notice that such a one-to-one mapping exists, because

$$|\mathcal{F}_{k'}| = \binom{k'}{k'/2} \geqslant 2^{k'}/\sqrt{2k'} \geqslant k.$$

For each node v, we compute the new color $\phi_{new}(v) \in \{1, 2, \dots, k'\}$ of v as follows. Let u be the parent of v. Since both $M(\phi_{old}(v))$ and $M(\phi_{old}(u))$ are subsets of size $k'/2$, and because $\phi_{old}(v) \neq \phi_{old}(u)$ and M is a one-to-one mapping, we know that $M(\phi_{old}(v)) \setminus M(\phi_{old}(u)) \neq \emptyset$. Let $\phi_{new}(v)$ be any arbitrary color in $M(\phi_{old}(v)) \setminus M(\phi_{old}(u))$. Since each node v gets a color $\phi_{new}(v) \in M(\phi_{old}(v)) \setminus M(\phi_{old}(u))$ that is not in the color set $M(\phi_{old}(u))$ of its parent, $\phi_{new}(v) \neq \phi_{new}(u) \in M(\phi_{old}(u))$. Hence, ϕ_{new} is a proper coloring. □

Remark 1.1. *Notice that in the above proof, the main property that we used is that none of the color-sets $M(i) \in \mathcal{F}_{k'}$ is contained in another $M(j) \in \mathcal{F}_{k'}$, for $i \neq j$. Generally, a family of sets such that none of them is contained in another*

[2]We assume this constant lower bound C_0 mainly to simplify our job and let us not worry about the rounding issues.

is called a Sperner Family. *In particular, ℓ-element subsets of a k'-element set form a Sperner family, the size of which is maximized by setting $\ell = \lfloor k'/2 \rfloor$, as we did above. More generally, Sperner's theorem shows that any Sperner family on a ground set of size k' has size at most $\binom{k'}{\lfloor k'/2 \rfloor}$ [Spe28]. See [Lub66] for a short and cute proof of Sperner's theorem, via a simple double counting.*

We can now iteratively apply the above method, abstracted in the statement of Lemma 1.5, to reduce the number of colors. After one round, we go from an initial n-coloring to a $(\log n + \log\log n/2 + 1)$-coloring. After one more round, we get to a coloring which has no more than $\log\log n + O(\log\log\log n)$ colors. More generally, after at most $\log^* n + O(1)$ repetitions, we get to a coloring with no more than C_0 colors, for a constant C_0. At this point, we cannot apply the above routine anymore. However, we can use an easier method that repeatedly uses two rounds to shave off one color, until arriving at a 3-coloring. We explain this next. Overall, we use $\log^* n + O(1)$ rounds to get a 3-coloring.

Lemma 1.6. *Given a k-coloring ϕ_{old} of a rooted tree where $k \geqslant 4$, in two rounds, we can compute a $(k-1)$-coloring ϕ_{new}.*

Proof. First, use one round where each node u sends its color $\phi_{old}(u)$ to its children. Then, let each node v set its temporary coloring $\phi'_{old}(v) = \phi_{old}(u)$, where u is the parent of v. For the root node r, this rule is not well-defined. But that is easy to fix. Define $\phi'_{old}(r) \in \{1,2,3\} \setminus \phi_{old}(r)$. Observe that ϕ'_{old} is a proper k-coloring, with the following nice additional property: for each node u, all of its children have the same color $\phi'_{old}(u)$.

Now, use another round where each node u sends its color $\phi'_{old}(u)$ to its children. Then, define the new color $\phi_{new}(v)$ as follows. For each node v such that $\phi'_{old}(v) \neq k$, let $\phi_{new}(v) = \phi'_{old}(v)$. For each node v such that $\phi'_{old}(v) = k$, let $\phi_{new}(v)$ be a color in $\{1,2,3\} \setminus \{\phi'_{old}(u), \phi_{old}(v)\}$.

Notice that since only nodes of color k are changing their color, these nodes are non-adjacent. Each of them switches to a color that is different than what is held by its parent and its children. Hence, the new coloring $\phi_{new}(v)$ is proper. □

Proof of Theorem 1.4. The proof follows by applying Lemma 1.5 for $\log^* n + O(1)$ iterations, until getting to a coloring with no more than $C_0 = O(1)$ colors, and then applying the method of Lemma 1.6 for $C_0 - 3 = O(1)$ iterations, until getting to a 3-coloring. □

Remark 1.2. *We can extend Theorem 1.4 to compute a coloring of an arbitrary graph with maximum degree Δ using $2^{O(\Delta)}$ colors, in $O(\log^* n)$ rounds. For that, we modify Lemma 1.5 as follows: for each node v, we view each of its up to Δ neighbors u as one "parent". We compute a new color similar to Lemma 1.5* *when comparing the old color of v with the old color of u. Then we take the Δ-tuple that lists all these Δ new colors. This way, in one iteration, we reduce the number of colors from k to $(O(\log k))^{\Delta}$, as there are $O(\log k)$ for each entry of the Δ-tuple. Repeating this for $O(\log^* n)$ iterations gets us to a coloring with $2^{O(\Delta)}$ colors. Notice that we can also turn this to a coloring with only $\Delta + 1$ colors, by spending $2^{O(\Delta)}$ extra rounds, where per round each node that has a color (strictly) great than all of its neighbors switches to a color in $\{1, \ldots, \Delta + 1\}$ not taken by its neighbors. In later sections, we see an algorithm for $\Delta + 1$ coloring with a much better dependency on Δ in its round complexity.*

1.2.2 3-Coloring Rooted Trees Needs $\frac{1}{2}\log^* n - O(1)$ Rounds

Theorem 1.7. *Any deterministic algorithm for 3-coloring n-node directed paths needs at least $\frac{\log^* n}{2} - 2$ rounds.*

For the sake of contradiction, suppose that there is an algorithm $\mathcal{A}$ that computes a 3-coloring of any n-node directed path in t rounds for $t < \frac{\log^* n}{2} - 2$. When running this algorithm for t rounds, any node v can see at most the k-neighborhood around itself for $k = 2t + 1$, that is, the vector of identifiers for the nodes up to t hops before itself and up to t hops after itself. Hence, if the algorithm $\mathcal{A}$ exists, there is a mapping from each such neighborhood to a color in $\{1, 2, 3\}$ such that neighborhoods that can be conceivably adjacent are mapped to different colors.

We next make this formal by a simple and abstract definition. For simplicity, we will consider only a restricted case of the problem where the identifiers are set monotonically increasing along the path. Notice this restriction only strengthens the lower bound, as it shows that even for this restricted case, there is no t-round algorithm for $t < \frac{\log^* n}{2} - 2$.

Definition 1.8. *We say B is a k-ary q-coloring if for any set of identifiers $1 \leqslant a_1 < a_2 < \cdots < a_k < a_{k+1} \leqslant n$, we have the following two properties:*

P1: $B(a_1, a_2, \ldots, a_k) \in \{1, 2, \ldots, q\}$,

P2: $B(a_1, a_2, \ldots, a_k) \neq B(a_2, \ldots, a_{k+1})$.

Observation 1.9. *If there exists a deterministic algorithm $\mathcal{A}$ for 3-coloring n-node directed paths in $t < \frac{\log^* n}{2} - 2$ rounds, then there exists a k-ary 3-coloring B, where $k = 2t + 1 < \log^* n - 3$.*

Proof. Suppose that such an algorithm $\mathcal{A}$ exists. We then produce a k-ary 3-coloring B by examining $\mathcal{A}$. For any set of identifiers $1 \leqslant a_1 < a_2 < \cdots < a_k \leqslant n$, define $B(a_1, a_2, \ldots, a_k)$ as follows. Simulate algorithm $\mathcal{A}$ on an imaginary directed path where a consecutive portion of the identifiers on the path are set equal to $a_1, a_2, \ldots, a_k$. Then, let $B(a_1, a_2, \ldots, a_k)$ be equal to the color in $\{1, 2, 3\}$ that the node a_{t+1} receives in this simulation.

We now argue that B as defined above is a k-ary 3-coloring. Property P1 holds trivially. We now argue that property P2 also holds. For the sake of contradiction, suppose that it does not, meaning that there exist a set of identifiers $1 \leqslant a_1 < a_2 < \cdots < a_k < a_{k+1} \leqslant n$ such that $B(a_1, a_2, \ldots, a_k) = B(a_2, \ldots, a_{k+1})$. Then, imagine running algorithm $\mathcal{A}$ on an imaginary directed path where a consecutive portion of identifiers are set equal to $a_1, a_2, \ldots, a_{2t+2}$. Then, since $B(a_1, a_2, \ldots, a_k) = B(a_2, \ldots, a_{k+1})$, the algorithm $\mathcal{A}$ assigns the same color to a_{t+1} and a_{t+2}. This is in contradiction with $\mathcal{A}$ being a 3-coloring algorithm. □

To prove Theorem 1.7, we show that a k-ary 3-coloring B where $k < \log^* n - 3$ cannot exist. The proof is based on the following two lemmas:

Lemma 1.10. *There is no* 1*-ary* q*-coloring with* $q < n$.

Proof. A 1-ary q-coloring requires that $B(a_1) \neq B(a_2)$, for any two identifiers $1 \leqslant a_1 < a_2 \leqslant n$. By the Pigeonhole principle, this needs $q \geqslant n$. □

Lemma 1.11. *If there is a* k*-ary* q*-coloring* B*, then there exists a* $(k-1)$*-ary* 2^q*-coloring* B'*.*

Proof. For any set of identifiers $1 \leqslant a_1 < a_2 < \cdots < a_{k-1} \leqslant n$, define $B'(a_1, a_2, \ldots, a_{k-1})$ to be the set of all possible colors $i \in \{1, \ldots, q\}$ for which $\exists a_k > a_{k-1}$ *such that* $B(a_1, a_2, \ldots, a_{k-1}, a_k) = i$.

Notice that B' is a subset of $\{1, \ldots, q\}$. Hence, it has 2^q possibilities, which means that B' has property P1 and it assigns each set of identifiers $1 \leqslant a_1 < a_2 < \cdots < a_{k-1} \leqslant n$ to a number in 2^q. Now we argue that B' also satisfies property P2.

For the sake of contradiction, suppose that there exist identifiers $1 \leqslant a_1 < a_2 < \cdots < a_k \leqslant n$ such that $B'(a_1, a_2, \ldots, a_{k-1}) = B'(a_2, a_3, \ldots, a_k)$. Let $q^* = B(a_1, a_2, \ldots, a_k) \in B'(a_1, a_2, \ldots, a_{k-1})$. Then, we must have $q^* \in B'(a_2, a_3, \ldots, a_k)$. Thus, $\exists a_{k+1} > a_k$ such that $q^* = B(a_2, a_3, \ldots, a_k, a_{k+1})$. But, in that case we would have $B(a_1, a_2, \ldots, a_k) = q^* = B(a_2, a_3, \ldots, a_k, a_{k+1})$, which is in contradiction with B being a k-ary q-coloring. Having reached at a contradiction by assuming that B' does not satisfy P2, we conclude that it actually does satisfy P2. Hence, B' is a $(k-1)$-ary 2^q-coloring. □

Proof of Theorem 1.7. For the sake of contradiction, suppose that there is an algorithm $\mathcal{A}$ that computes a 3-coloring of any n-node directed path in t rounds for $t < \frac{\log^* n}{2} - 2$. As stated in Observation 1.9, if there exists an algorithm $\mathcal{A}$ that computes a 3-coloring of any n-node directed path in t rounds for $t < \frac{\log^* n}{2} - 2$, then there exists a k-ary 3-coloring B, where

$k = 2t + 1 < \log^* n - 3$. Using one iteration of Lemma 1.11, we would get that there exists a $(k-1)$-ary 8-coloring. Another iteration would imply that there exists a $(k-2)$-ary 2^8-coloring. Repeating this, after $k < \log^* n - 3$ iterations, we would get a 1-ary coloring with less than n colors. However, this is in contradiction with Lemma 1.10. Hence, such an algorithm $\mathcal{A}$ cannot exist. □

An Alternative Lower Bound Proof Via Ramsey Theory:
Let us first briefly recall the basics of Ramsey Theory. The simplest case of Ramsey's theorem says that for any ℓ, there exists a number $R(\ell)$ such that for any $n \geqslant R(\ell)$, if we color the edges of the n-node complete graph K_n with two colors, there exists a monochromatic clique of size ℓ in it, that is, a set of ℓ vertices such that all of the edges between them have the same color. A simple example is that among any group of at least $6 = R(3)$ people, there are either at least 3 of them which are friends, or at least 3 of them no two of which are friends.

A similar statement is true in hypergraphs. Of particular interest for our case is coloring hyperedges of a complete n-vertex hypergraph of rank k, that is, the hypergraph where every subset of size k of the vertices defines one hyperedge. By Ramsey theory, it is known that there exists an n_0 such that, if $n \geqslant n_0$, for any way of coloring hyperedges of the complete n-vertex hypergraph of rank k with 3 colors, there would be a monochromatic clique of size $k+1$. That is, there would be a set of $k+1$ vertices $a_1, \ldots, a_{k+1}$ in $\{1, \ldots, n\}$ such that all of their $\binom{k+1}{k} = k+1$ subsets with cardinality k have the same color.

In particular, consider an arbitrary k-ary coloring B, and let B define the colors of the hyperedges $\{a_1, \ldots, a_k\}$ when $1 \leqslant a_1 < a_2 < \cdots < a_k \leqslant n$. By Ramsey's theorem, we would get the following: there exist vertices $1 \leqslant a_1 < a_2 < \cdots < a_k < a_{k+1} \leqslant n$ such that B assigns the same color to hyperedges $\{a_1, \ldots, a_k\}$ and $\{a_2, \ldots, a_{k+1}\}$. But this is in contradiction with the property P2 of B being a k-ary coloring. The value of n_0 that follows from Ramsey theory is such that $k = O(\log^* n_0)$. In other words, Ramsey's theorem rules out $o(\log^* n)$-round 3-coloring algorithms for directed paths. See [CFS10] for more on hypergraph Ramsey numbers.

1.3 Coloring Unrooted Trees

In the previous sections, we saw that on a rooted tree, where each node knows its parent, a 3-coloring can be computed distributedly in $O(\log^* n)$ rounds. Moreover, we proved that this round complexity is optimal. This $O(\log^* n)$-round algorithm for rooted trees heavily relies on each node knowing its parent.

We next prove that no such result is possible in unrooted trees, when nodes do not know which neighbor is their parent. More concretely, we prove that any deterministic algorithm for coloring unrooted trees that runs in $o(\log_\Delta n)$ rounds must use at least $\Omega(\Delta/\log\Delta)$ colors. Moreover, we complement this by showing that given $O(\log n)$ rounds, we can compute a 3-coloring of any n-node tree.

1.3.1 The Lower Bound

Theorem 1.12. *Any (deterministic) distributed algorithm $\mathcal{A}$ that colors n-node trees with maximum degree Δ using less than $o(\Delta/\log\Delta)$ colors has round complexity at least $\Omega(\log_\Delta n)$.*

To prove the claimed lower bound, we will use a graph-theoretic result about the existence of certain graphs. The proof of this lemma is based on a *probabilistic method* argument, but we do not cover it in this lecture. Before stating the properties of this graph, we recall that the *girth* of a graph is the length of the shortest cycle.

Fact 1.13 (Bollobas [Bol78]). *There exists an infinite family of n-node graphs H^* where all nodes have degree Δ, with girth $g(H^*) = \Omega(\log_\Delta n)$ and chromatic number $\chi(H^*) = \Omega(\Delta/\log\Delta)$.*

Remark 1.3. *We note that this lower bound on the chromatic number is asymptotically tight, because high-girth graphs, and more generally triangle-free graphs, with maximum degree Δ have chromatic number $O(\Delta/\log\Delta)$ [Kim95, Jam11, PS15].*

Proof of Theorem 1.12. For the sake of contradiction, suppose that there exists a deterministic distributed algorithm $\mathcal{A}$ that computes a $o(\Delta/\log\Delta)$-coloring of any n-node tree with maximum degree Δ, in $o(\log_\Delta n)$ rounds.

We run $\mathcal{A}$ on the graph H^*, stated in Fact 1.13. Notice that H^* is not a tree. However, since $g(H^*) = \Omega(\log_\Delta n)$, within the $o(\log_\Delta n)$ rounds of the algorithm, no one will notice! In particular, since $g(H^*) = \Omega(\log_\Delta n)$, for any node v, the subgraph T_v of H^* induced by nodes within distance $o(\log_\Delta n)$ of v is a tree (why?). Thus, within the $o(\log_\Delta n)$ rounds of the algorithm, node v will think that the algorithm $\mathcal{A}$ is being run on T_v and will not realize that the algorithm is being run on a non-tree graph H^*. Similarly, none of the nodes will recognize that we are not on a tree. Hence, each node v will compute an output as if the algorithm $\mathcal{A}$ was being run on its local tree T_v. This must produce a valid coloring of H^* with $o(\Delta/\log\Delta)$ colors. That is because if the algorithm creates two neighbors v and u with the same color, then running the algorithm on the tree T_v would also produce a non-valid color. However, the fact that $\mathcal{A}$ is able to compute a $o(\Delta/\log\Delta)$-coloring of H^* is in contradiction with the fact that $\chi(H^*) = \Omega(\Delta/\log\Delta)$. □

1.3.2 The Upper Bound

Theorem 1.14. *There is a deterministic distributed algorithm that computes a 3-coloring of any n-node tree in $O(\log n)$ rounds.*

The Algorithm for Coloring Unrooted Trees, Step 1 We first perform an iterated peeling process, on the given tree $T = (V, E)$. Let $T_1 = T$ and let layer L_1 be the set of all vertices of T_1 whose degree in T_1 is at most 2. Then, let $T_2 = T_1 \setminus L_1$ be the forest obtained by removing from T_1 all the L_1 vertices. Then, define layer L_2 be the set of all vertices of T_2 whose degree in T_2 is at most 2. Then, define $T_3 = T_2 \setminus L_2$ similar to before. More generally, each T_{i+1} is defined as $T_{i+1} = T_i \setminus L_i$, which is the forest obtained by removing from T_i all the layer L_i vertices, and then layer L_{i+1} is defined to be the vertices that have degree at most 2 in T_{i+1}.

Lemma 1.15. *The process terminates in $O(\log n)$ iterations, meaning that V gets decomposed into disjoint sets $L_1, L_2, \ldots, L_\ell$ for some $\ell = O(\log n)$.*

Proof. T_i has at most $|T_i| - 1$ edges, since it is a forest. Hence, the number of vertices of T_i that have degree at least 3 is at most $(2|T_i| - 1)/3$. Hence, at least $1/3$ of the nodes of T_i are put in L_i. This means in each iteration the number of nodes reduces by a $2/3$-factor, which implies that we are done within $\ell = \log_{3/2} n$ iterations. $\square$

The Algorithm for Coloring Unrooted Trees, Step 2 Now, we color each of the subgraphs $T[L_i]$ independently using 3 colors, in $O(\log^* n)$ rounds. Notice that since $T[L_i]$ has maximum degree at most 2, this is doable for instance using the algorithm from Theorem 1.4 (concretely its extension outlined in Remark 1.2) or alternatively using the algorithm that we will later see in Theorem 1.22. We use these colors of the graphs $T[L_i]$ mainly as a *schedule-color*, for computing the final output coloring of the vertices.

The Algorithm for Coloring Unrooted Trees, Step 3 We process the graph by going through the layers L_ℓ to L_1, spending 3 rounds on each. Each time, we make sure that we have a valid *final-coloring* of the graph $T[\cup_{j=i}^{\ell} L_i]$ with 3 colors, for decreasing value of i.

Suppose we have an arbitrary final-coloring of $T[\cup_{j=i+1}^{\ell} L_j]$ already, with 3 colors. How do we compute a 3-coloring for vertices of L_i in a manner that does not create a violation with the colors of vertices of $T[\cup_{j=i+1}^{\ell} L_j]$?

This can be done easily using our usual trick of applying one coloring as a schedule for computing another coloring. In particular, we will solve this part of the problem in 3 rounds. We go through the 3 schedule-colors

$q \in \{1, 2, 3\}$ of $T[L_i]$, one by one, each time picking a *final-color* in $\{1, 2, 3\}$ for all the vertices in L_i with schedule color q.

1.4 Deterministic Coloring of General Graphs

1.4.1 Take 1: Linial's Coloring Algorithm

In the previous section, we discussed distributed LOCAL algorithms for coloring oriented trees. In this section, we start the study of LOCAL coloring algorithms for general graphs. Throughout, the ultimate goal would be to obtain $(\Delta + 1)$-coloring of the graphs — that is, an assignment of colors $\{1, 2, \ldots, \Delta + 1\}$ to vertices such that no two adjacent vertices receive the same color — where Δ denotes the maximum degree. Notice that by a simple greedy argument, each graph with maximum degree at most Δ has a $(\Delta + 1)$-coloring: color vertices one by one, each time picking a color which is not chosen by the already-colored neighbors. However, this greedy argument does not lead to an efficient LOCAL procedure for finding such a coloring[3].

In this section, we start with presenting an $O(\log^* n)$-round algorithm that computes a $O(\Delta^2)$ coloring. This algorithm is known as Linial's coloring algorithm [Lin87, Lin92]. In Section 1.4.2, we see how to transform this coloring into a $(\Delta + 1)$-coloring.

Theorem 1.16. *There is a deterministic distributed algorithm in the* LOCAL *model that colors any n-node graph G with maximum degree Δ using $O(\Delta^2)$ colors, in $O(\log^* n)$ rounds.*

Outline. The core piece of the algorithm is a single-round color reduction method, as we will describe in Lemma 1.17. That will allows us to transform any given coloring with some k colors to some other coloring with a much smaller number $k' \ll k$ of colors. Then, by repeated applications of this single-round color reduction, we obtain the coloring algorithm as claimed in Theorem 1.16.

Lemma 1.17. *Given a k-coloring ϕ_{old} of a graph with maximum degree Δ, in a single round, we can compute a k'-coloring ϕ_{new}, for $k' = O(\Delta^2 \log k)$. Furthermore, if $k \leqslant \Delta^3$, then the bound can be improved to $k' = O(\Delta^2)$.*

We will come back to proving this lemma. Let us first see how by using it we can immediately obtain Theorem 1.16.

[3]The straightforward transformation of this greedy approach to the LOCAL model would be an algorithm that may need $\Omega(n)$ rounds.

Proof of Theorem 1.16. The proof is via iterative applications of Lemma 1.17. We start with the initial numbering of the vertices as a straightforward n-coloring. With one application of Lemma 1.17, we transform this into a $O(\Delta^2 \log n)$ coloring. With another application, we get a coloring with $O(\Delta^2(\log \Delta + \log\log n))$ colors. With another application, we get a coloring with $O(\Delta^2(\log \Delta + \log\log\log n))$ colors. After $O(\log^* n)$ applications, we get a coloring with $O(\Delta^2 \log \Delta)$ colors (why[4]?). At this point, we use one extra iteration, based on the second part of Lemma 1.17, which gets us to an $O(\Delta^2)$-coloring. □

Single-Round Color Reduction

We now go back to Lemma 1.17 and explain its color reduction method. We note that this single-round color reduction method can be seen as a much more general variant of the single-round color reduction that we discussed in Lemma 1.5 for coloring rooted trees. The difference is that here, each node has to ensure that the color that it picks is different than all of its neighbors, and not just its parents.

The key concept in our single-round color reduction is a combinatorial notion called cover free families, as we will define next.

Definition 1.18. *(**Cover free families**) Given a ground set $\{1, 2, \ldots, k'\}$, a family of sets $S_1, S_2, \ldots, S_k \subseteq \{1, 2, \ldots, k'\}$ is called a Δ-cover free family if for each set of indices $i_0, i_1, i_2, \ldots, i_\Delta \in \{1, 2, \ldots, k\}$, we have $S_{i_0} \setminus \left(\cup_{j=1}^{\Delta} S_{i_j}\right) \neq \emptyset$. That is, if no set in the family is a subset of the union of Δ other sets.*

We comment that cover free families can be seen as a generalization of Sperner families (as mentioned in Remark 1.1 and used in the single-round color reduction of Lemma 1.5 for rooted trees): a Sperner family is simply a 1-cover free family, i.e., no set is a subset of any other set.

Using cover free families for color reduction. We use cover free families for color reduction in the obvious way: consider an old coloring ϕ_{old} with k colors and suppose we want a new coloring ϕ_{new} with k' colors. Each node v of old color $\phi_{old}(v) = q$ for $q \in \{1, \ldots, k\}$ will use the set $S_q \subseteq \{1, \ldots, k'\}$ in the cover free family as its *color-set*, i.e., its list of possible colors. Then, it sets its new color $\phi_{new}(v) = q'$ where $q' \in S_q$ is such that q' is not in the color-set of any of the neighbors. Such a color q' is promised to exist, by the definition of cover free families.

As clear from the above outline, we would like to have k' as small as possible, as a function of k and Δ. This would allow us to reduce the

[4]If this is not clear, please ask during the exercise sessions.

number of colors faster. In the following, we prove the existence of Δ-cover free families with a small enough ground set size k'. In particular, Lemma 1.19 achieves $k' = O(\Delta^2 \log k)$ and Lemma 1.20 shows that this bound can be improved to $k' = O(\Delta^2)$, if $k \leqslant \Delta^3$. Toward the end of this subsection, we provide the formal proof that these imply Lemma 1.17.

Lemma 1.19. *(**Existence of cover free families**) For any k and Δ, there exists a Δ-cover free family of size k on a ground set of size $k' = O(\Delta^2 \log k)$.*

Proof. We use the probabilistic method [AS04] to argue that there exists a Δ-cover free family of size k on a ground set of size $k' = O(\Delta^2 \log k)$. Let $k' = C\Delta^2 \log k$ for a sufficiently large constant $C \geqslant 2$. For each $i \in \{1, 2, \ldots, k\}$, define each set $S_i \subset \{1, 2, \ldots, k'\}$ randomly by including each element $q \in \{1, 2, \ldots, k'\}$ in S_i with probability $p = 1/\Delta$. We argue that this random construction is indeed a Δ-cover free family, with probability close to 1. Therefore, such a cover free family exists.

First, consider an arbitrary set of indices $i_0, i_1, i_2, \ldots, i_\Delta \in \{1, 2, \ldots, k\}$. We would like to argue that $S_{i_0} \setminus \big(\cup_{j=1}^{\Delta} S_{i_j}\big) \neq \emptyset$. For each element $q \in \{1, 2, \ldots, k'\}$, the probability that $q \in S_{i_0} \setminus \big(\cup_{j=1}^{\Delta} S_{i_j}\big)$ is at exactly $\frac{1}{\Delta}(1 - \frac{1}{\Delta})^{\Delta} \geqslant \frac{1}{4\Delta}$. Hence, the probability that there is no such element q that is in $S_{i_0} \setminus \big(\cup_{j=1}^{\Delta} S_{i_j}\big)$ is at most $(1 - \frac{1}{4\Delta})^{k'} \leqslant \exp(-C\Delta \log k/4)$. This is an upper bound on the probability that for a given set of indices $i_0, i_1, i_2, \ldots, i_\Delta \in \{1, 2, \ldots, k\}$, the respective sets violate the cover-freeness property that $S_{i_0} \setminus \big(\cup_{j=1}^{\Delta} S_{i_j}\big) \neq \emptyset$.

There are $k\binom{k-1}{\Delta}$ way to choose such a set of indices $i_0, i_1, i_2, \ldots, i_\Delta \in \{1, 2, \ldots, k\}$, k ways for choosing the central index i_0 and at most $(k-1)^\Delta$ ways for choosing the indices $i_1, i_2, \ldots, i_\Delta$. Hence, by a union bound over all these choices, the probability that the construction fails is at most

$$\begin{aligned} k(k-1)^\Delta \cdot \exp(-C\Delta \log k/4) &= \exp\big(\log k + \Delta(\log(k-1)) - C\Delta \log k/4\big) \\ &\leqslant \exp(-C\Delta \log k/8) \ll 1, \end{aligned}$$

for a sufficiently large constant C. That is, the random construction succeeds to provide us with a valid Δ-cover free family with a positive probability, and in fact with a probability close to 1. Hence, such a Δ-cover free family exists. □

Lemma 1.20. *For any k and $\Delta \geqslant k^{1/3}$, there exists a Δ-cover free family of size k on a ground set of size $k' = O(\Delta^2)$.*

Proof. Here, we use an algebraic proof based on low-degree polynomials. Let q be a prime number that is in $[3\Delta, 6\Delta]$. Notice that such a prime number exists by Bertrand's postulate (also known as Bertrand-Chebyshev

Theorem). Let $\mathbb{F}_q$ denote the prime field[5] of order q (i.e., integers modulo q). For each $i \in \{1, 2, \ldots, k\}$, associate with set S_i — to be constructed — a distinct degree $d = 2$ polynomial $g_i : \mathbb{F}_q \to \mathbb{F}_q$ over $\mathbb{F}_q$. Notice that there are $q^{d+1} > \Delta^3 \geqslant k$ such polynomials and hence such an association is possible. Let S_i be the set of all evaluation points of g_i, that is, let $S_i = \{(a, g_i(a)) \,|\, a \in \mathbb{F}_q\}$. These are subsets of the $k' = q^2$ cardinality set $\mathbb{F}_q \times \mathbb{F}_q$. Notice two key properties:

(A) for each $i \in \{1, 2, \ldots, k\}$, we have $|S_i| = q$.

(B) for each $i, i' \in \{1, 2, \ldots, k\}$ such that $i \neq i'$, we have $|S_i \cap S_{i'}| \leqslant d$.

The latter property holds because, in every intersection point, the degree d polynomial $g_i - g_{i'}$ evaluates to zero, and each degree d polynomial has at most d zeros. Now, the Δ cover-freeness property follows trivially from (A) and (B), because for any set of indices $i_0, i_1, i_2, \ldots, i_\Delta \in \{1, 2, \ldots, k\}$, we have

$$\begin{aligned} |S_{i_0} \setminus (\cup_{j=1}^{\Delta} S_{i_j})| \geqslant |S_{i_0}| &- \sum_{j=1}^{\Delta} |S_{i_0} \cap S_{i_j}| \\ \geqslant q &- \Delta \cdot d = q - 2\Delta \geqslant \Delta \geqslant 1. \end{aligned}$$

□

Remark 1.4. *One can easily generalize the construction of Lemma 1.20, by taking higher-degree polynomials, to a ground set of size $k' = O(\Delta^2 \log_\Delta^2 k)$, where no assumption on the relation between k and Δ would be needed.*

Proof Sketch of Lemma 1.17. Follows from the existence of cover free families as proven in Lemma 1.19 and Lemma 1.20. Namely, each node v of old color $\phi_{old}(v) = q$ for $q \in \{1, \ldots, k\}$ will use the set $S_q \subseteq \{1, \ldots, k'\}$ in the cover free family as its *color-set*. Then, it sets its new color $\phi_{new}(v) = q'$ for a $q' \in S_q$ such that q' is not in the color-set of any of the neighbors. By the definition of the cover free families, and given that ϕ_{old} was a proper coloring, we are guaranteed that such a color q' exists. By the choice of q', the coloring ϕ_{new} is also a proper coloring. □

1.4.2 Take 2: Kuhn-Wattenhofer Coloring Algorithm

In the previous section, we saw an $O(\log^* n)$-round algorithm for computing a $O(\Delta^2)$-coloring. In this section, we explain how to transform this into a $(\Delta+1)$-coloring. We will first see a very basic algorithm that performs this transformation in $O(\Delta^2)$ rounds. Then, we see how with the addition of a small but clever idea of [KW06], this transformation can be performed in $O(\Delta \log \Delta)$ rounds. As the end result, we get an $O(\Delta \log \Delta + \log^* n)$-round algorithm for computing a $(\Delta+1)$-coloring.

[5] See https://en.wikipedia.org/wiki/Finite_field

Warm up: One-By-One Color Reduction

Lemma 1.21. *Given a k-coloring ϕ_{old} of a graph with maximum degree Δ where $k \geqslant \Delta + 2$, in a single round, we can compute a $(k-1)$-coloring ϕ_{new}.*

Proof. For each node v such that $\phi_{old}(v) \neq k$, set $\phi_{new}(v) = \phi_{old}(v)$. For each node v such that $\phi_{old}(v) = k$, let node v set its new color $\phi_{new}(v)$ to be a color $q \in \{1, 2, \dots, \Delta + 1\}$ such that q is not taken by any of the neighbors of u. Such a color q exists, because v has at most Δ neighbors. The resulting new coloring ϕ_{new} is a proper coloring. □

Theorem 1.22. *There is a deterministic distributed algorithm in the* LOCAL *model that colors any n-node graph G with maximum degree Δ using $\Delta + 1$ colors, in $O(\Delta^2 + \log^* n)$ rounds.*

Proof. First, compute an $O(\Delta^2)$-coloring in $O(\log^* n)$ rounds using the algorithm of Theorem 1.16. Then, apply the one-by-one color reduction of Lemma 1.21 for $O(\Delta^2)$ rounds, until getting to a $(\Delta + 1)$-coloring. □

Parallelized Color Reduction

Lemma 1.23. *Given a k-coloring ϕ_{old} of a graph with maximum degree Δ where $k \geqslant \Delta + 2$, in $O(\Delta \log(\frac{k}{\Delta+1}))$ rounds, we can compute a $(\Delta + 1)$-coloring ϕ_{new}.*

Proof. If $k \leqslant 2\Delta + 1$, the lemma follows immediately from applying the one-by-one color reduction of Lemma 1.21 for $k - (\Delta + 1)$ iterations. Suppose that $k \geqslant 2\Delta + 2$. Bucketize the colors $\{1, 2, \dots, k\}$ into $\lfloor \frac{k}{2\Delta+2} \rfloor$ buckets, each of size exactly $2\Delta + 2$, except for one last bucket which may have size between $2\Delta + 2$ to $4\Delta + 3$. We can perform color reductions in all buckets in parallel (why?). In particular, using at most $3\Delta + 2$ iterations of one-by-one color reduction of Lemma 1.21, we can recolor nodes of each bucket using at most $\Delta + 1$ colors. Considering all buckets, we now have at most $(\Delta + 1)\lfloor \frac{k}{2\Delta+2} \rfloor \leqslant k/2$ colors. Hence, we managed to reduce the number of colors by a 2 factor, in just $O(\Delta)$ rounds. Repeating this procedure for $\lceil \log(\frac{k}{\Delta+1}) \rceil$ iterations gets us to a coloring with $\Delta + 1$ colors. The round complexity of this method is $O(\Delta \log(\frac{k}{\Delta+1}))$, because we have $\lceil \log(\frac{k}{\Delta+1}) \rceil$ iterations and each iteration takes $O(\Delta)$ rounds. □

Theorem 1.24. *There is a deterministic distributed algorithm in the* LOCAL *model that colors any n-node graph G with maximum degree Δ using $\Delta + 1$ colors, in $O(\Delta \log \Delta + \log^* n)$ rounds.*

Proof. First, compute an $O(\Delta^2)$-coloring in $O(\log^* n)$ rounds using the algorithm of Theorem 1.16. Then, apply the parallelized color reduction of Lemma 1.23 to transform this into a $(\Delta + 1)$-coloring, in $O(\Delta \log \Delta)$ additional rounds. □

1.4.3 Take 3: Kuhn's Algorithm via Defective Coloring

In the previous section, we saw an algorithm that computes a $(\Delta+1)$-coloring in $O(\Delta \log \Delta + \log^* n)$ rounds. We now present an algorithm that improves this round complexity to $O(\Delta + \log^* n)$ rounds, based on a *Defective Coloring* method of Kuhn [Kuh09].

Theorem 1.25. *There is a deterministic distributed algorithm in the* LOCAL *model that colors any n-node graph G with maximum degree Δ using $\Delta+1$ colors, in $O(\Delta + \log^* n)$ rounds.*

It is worth noting that this linear-in-Δ round complexity remained as the state of the art, and it looked as if it might be the best possible for deterministic algorithms[6], until 2015. But then came a breakthrough of Barenboim [Bar15] which computed a $(\Delta+1)$-coloring in $O(\Delta^{3/4} \log \Delta + \log^* n)$ rounds. This was followed by a beautiful work of Fraigniaud, Heinrich, and Kosowski [FHK16], which improved the round complexity of $(\Delta+1)$-coloring further to $O(\Delta^{1/2} \log \Delta^{2.5} + \log^* n)$ rounds. We will not cover these recent advances in our lectures, but the papers should be already accessible and easy to follow, given what we have covered so far. What is the optimal round complexity for deterministic $(\Delta+1)$-coloring algorithms remains an intriguing and long-standing open problem – an ultimate goal would be to deterministically compute a $(\Delta+1)$ coloring in $\operatorname{poly}\log(n)$ rounds.

Definition 1.26. *For a graph $G = (V, E)$, a color assignment $\phi : V \to \{1, 2, \dots, k\}$ is called a d-defective k-coloring if the following property is satisfied: for each color $q \in \{1, 2, \dots, k\}$, the subgraph of G induced by vertices of color q has maximum degree at most d. In other words, in a d-defective coloring, each node v has at most d neighbors that have the same color as v.*

Notice that a standard proper k-coloring — where no two adjacent nodes have the same color — is simply a 0-defective k-coloring.

Lemma 1.27. *Given a d-defective k-coloring ϕ_{old} of a graph with maximum degree Δ, in a single round, we can compute a d'-defective k'-coloring ϕ_{new}, for $k' = O\left(\left(\frac{\Delta-d}{d'-d+1}\right)^2 \log k\right)$.*

Proof. Proof to be added here. See pages 10 to 13 of this handwritten lecture note[7], for now. □

[6]Randomized algorithms have a very different story: We will see a simple $O(\log n)$-round randomized $\Delta+1$ coloring algorithm in the next sections, and we will also touch upon further improvements on the randomized track.

[7]http://people.csail.mit.edu/ghaffari/DGA14/Notes/L02.pdf

In a sense, this color reduction reduces the number of colors significantly, while increasing the defect only slightly.

Proof of Theorem 1.25. First, compute an 0-defective $C\Delta^2$-coloring, in $O(\log^* n)$-rounds, using the algorithm of Theorem 1.16. Here, C is a sufficiently large constant, as needed in Theorem 1.16. We will now see how to improve this to $\Delta+1$ colors.

The method is recursive. Let $T(\Delta)$ denote the complexity of $(\Delta+1)$-coloring graphs with maximum degree Δ, given the initial $O(\Delta^2)$-coloring. to perform a recursive method, we would like to decompose the graph into a few subgraphs of degree at most $\Delta/2$ and proceed recursively. In the following, we explain how to do this, using defective coloring as a tool.

We start with an $(C\Delta^2)$-coloring, as computed before, for a large enough constant $C > 0$. Then, we use one iteration of Lemma 1.27 to transform this into a $(\frac{\Delta}{\log \Delta})$-defective $O(\log^3 \Delta)$-coloring. Then, use another iteration of Lemma 1.27 to transform this into a $(\frac{\Delta}{\log\log \Delta})$-defective $O(\log^3 \log \Delta)$-coloring. One more iteration gets us to a $(\frac{\Delta}{\log\log\log \Delta})$-defective $O(\log^3 \log\log \Delta)$-coloring. After $O(\log^* \Delta)$ iterations, we get a $(\frac{\Delta}{2})$-defective k''-coloring for $k'' = O(1)$.

Now, each of these k'' color classes induces a subgraph with maximum degree $\Delta/2$. That is, we have decomposed the graph G into $O(1)$ disjoint subgraphs $G_1, G_2, \dots, G_{O(1)}$, each with maximum degree at most $\Delta/2$. Hence, by recursion, we can color each of them using $\Delta/2+1$ colors, all in parallel, in $T(\Delta/2)$ rounds. Formally, to be able to invoke the recursion, we should provide to each G_i an initial coloring with $C(\Delta/2)^2$ coloring. Notice that this can be computes easily in at most $O(\log^* n)$ time, using Linial's recoloring method as covered in Theorem 1.16. This allows us to invoke the recursive coloring procedure, and get a $\Delta/2+1$ coloring for each G_i. When paired with the corresponding subgraph G_i index i, these color form an $\Delta/2 \cdot O(1) = O(\Delta)$ coloring of the whole graph. This can be transformed into a $\Delta+1$ coloring, in $O(\Delta)$ extra rounds, using the one-by-one color reduction method of Lemma 1.21.

As a result, we get the recursion

$$T(\Delta) = O(\log^* \Delta) + T(\Delta/2) + O(\Delta).$$

Recalling the Master theorem for recursions [CLRS01], we easily see that the answer of this recursion is $T(\Delta) = O(\Delta)$. Hence, including the initial $O(\log^* n)$-rounds, this is an overall round complexity of $O(\Delta + \log^* n)$. □

1.5 Network Decomposition

In the previous sections, we zoomed in on one particular problem, graph coloring, and we discussed a number of algorithms for it. In this section, we will discuss a method that is far more general and can be used for a wide range of *local* problems. The key concept in our discussion will be *network decompositions* first introduced by Awerbuch et al. [ALGP89], also known as *low-diameter graph decomposition* [LS91].

1.5.1 Definition and Applications

Let us start with defining the concept of network decompositions:

Definition 1.28. *(Weak Diameter Network Decomposition) Given a graph* $G = (V, E)$*, a* $(\mathcal{C}, \mathcal{D})$ *weak diameter network decomposition of* G *is a partition of* G *into vertex-disjoint graphs* G_1, G_2, ..., $G_{\mathcal{C}}$ *such that for each* $i \in \{1, 2, \dots, \mathcal{C}\}$*, we have the following property: the graph* G_i *is made of a number of vertex-disjoint and mutually non-adjacent clusters* X_1, X_2, ..., X_ℓ*, where each two vertices* $v, u \in X_j$ *have distance at most* D *in graph* G*. We note that we do not bound the number* ℓ*. We refer to each subgraph* G_i *as one* block *of this network decomposition.*

Definition 1.29. *(Strong Diameter Network Decomposition) Given a graph* $G = (V, E)$*, a* $(\mathcal{C}, \mathcal{D})$ *strong diameter network decomposition of* G *is a partition of* G *into vertex-disjoint graphs* G_1, G_2, ..., $G_{\mathcal{C}}$ *such that for each* $i \in \{1, 2, \dots, \mathcal{C}\}$*, we have the following property: each connected component of* G_i *has diameter at most* D.

Notice that a strong diameter network decomposition is also a weak diameter network decomposition. For

Network decompositions can be used to solve a wide range of *local* problems. To see the general method in a concrete manner, let us go back to our beloved $(\Delta + 1)$-coloring problem.

Theorem 1.30. *Provided an* $(\mathcal{C}, \mathcal{D})$ *weak-diameter network decomposition of a graph* G*, we can compute a* $\Delta + 1$ *coloring of* G *in* $O(CD)$ *rounds.*

Proof. We will color graphs G_1, G_2, ..., $G_{\mathcal{C}}$ one by one, each time considering the coloring assigned to the previous subgraphs. Suppose that vertices of graphs graphs G_1, G_2, ..., G_i are already colored using colors in $\{1, 2, \dots, \Delta + 1\}$. We explain how to color G_{i+1} in $O(D)$ rounds. Consider the clusters X_1, X_2, ..., X_ℓ of G_{i+1} and notice their two properties: (1) they are mutually non-adjacent, (2) for each cluster X_j, its vertices are within distance D of each other (where distances are according to the base graph G). For each cluster X_j, let node $v_j \in X_j$ who has the maximum identifier among nodes of X_j be the leader of X_j. Notice that leaders of clusters $X_1, X_2, \ldots, X_\ell$ can be identified in $O(D)$ rounds (why?). Then, let v_j aggregate the topology of the subgraph induced by X_j as well as the colors assigned to nodes adjacent to X_j in the previous graphs $G_1, G_2, \ldots, G_i$. This again can be done in $O(D)$ rounds, thanks to the fact that all the relevant information is within distance $D+1$ of v_j. Once this information is gathered, node v_j can compute a $(\Delta+1)$-coloring for vertices of X_j, while taking into account the colors of neighboring nodes of previous graphs, using a simple greedy procedure. Then, node v_j can report back these colors to nodes of X_j. This will happen for all the clusters $X_1, X_2, \ldots, X_\ell$ in parallel, thanks to the fact that they are non-adjacent and thus, their coloring choices does not interfere with each other. $\square$

In the next subsections, we first discuss a polylogarithmic-time randomized construction for network decompositions, and then we present two deterministic constructions for it. The first deterministic construction is a classic result from 1990s and it only achieves a subpolynomial time complexity, but not a polylogarithmic time. The second deterministic construction is a very recent result and it gives the first polylogarithmic time deterministic algorithm for network decomposition (and hence for a wide range of other central problems in the area).

1.5.2 Randomized Construction

Theorem 1.31. *There is a randomized* LOCAL *algorithm that computes a $(\mathcal{C}, \mathcal{D})$ weak-diameter network decomposition of any n-node graph G, for $\mathcal{C} = O(\log n)$ and $\mathcal{D} = O(\log n)$, in $O(\log^2 n)$ rounds, with high probability*[8].

We remark that, as we will see in Exercise 1.10, the round complexity of this construction can be improved to $O(\log n)$ rounds. On the other hand, as we see in Exercise 1.11, the two key parameters $\mathcal{C}$ and $\mathcal{D}$ are nearly optimal and one cannot improve them simultaneously and significantly.

Network Decomposition Algorithm: Suppose that we have already computed subgraphs $G_1, \ldots, G_i$ so far. We now explain how to compute a subgraph $G_{i+1} \subseteq G \setminus \left(\cup_{j=1}^{i} G_j\right)$, in $O(\log n)$ rounds, which would satisfy the properties of one block of a weak diameter network decomposition.

Let each node v pick a random radius r_u from an geometric distribution with parameter ε, for a desired (free parameter) constant $\varepsilon \in (0,1)$. That is, for each integer $y \geqslant 1$, we have $\Pr[r_u = y] = \varepsilon(1-\varepsilon)^{y-1}$. We will think of the

[8]Throughout, we will use the phrase *with high probability to indicate that an event happens with probability at least $1 - \frac{1}{n^c}$, for a desirably large but fixed constant $c \geqslant 2$.*

vertices within distance r_u of u as the *ball of node* u. Now for each node v, let $\mathtt{Center}(v)$ be the node u^* among nodes u such that $\mathtt{dist}_G(u, v) \leqslant r_u$ that has the smallest identifier. That is, $\mathtt{Center}(v) = u^*$ is the smallest-identifier node whose ball contains v. Define the clusters of G_i by letting all nodes with the same center define one cluster, and then discarding nodes who are at the boundary of their cluster. That is, any node v for which $\mathtt{dist}_G(v, u) = r_u$ where $u = \mathtt{Center}(v)$ remains unclustered.

There are two properties to prove: one that the clusters have low diameter, and second, that after $\mathcal{C}$ iterations, all nodes are clustered. In the following two lemmas, we argue that with high probability, each cluster has diameter $O(\log n/\varepsilon)$ and after $\mathcal{C} = O(\log_{1/\varepsilon} n)$ iterations, all nodes are clustered.

Lemma 1.32. *With high probability, the maximum cluster diameter is at most* $O(\log n/\varepsilon)$*. Hence, this clustering can be computed in* $O(\log n/\varepsilon)$ *rounds, with high probability.*

Proof. The proof is simple and is left as an exercise. □

Lemma 1.33. *For each node* v*, the probability that* v *is not clustered — that* v *is on the boundary of its supposed cluster and thus it gets discarded — is at most* ε*.*

Proof. Notice that

$$\begin{aligned} &\Pr[v \textit{ is not clustered }] = \\ \sum_{u \in V} &\Pr[v \textit{ is not clustered } \mid \mathtt{Center}(v) = u] \cdot \Pr[\mathtt{Center}(v) = u] \end{aligned}$$

For each vertex u, let $\mathtt{before}(u)$ denote the set of all vertices whose identifier is less than that of u. Define the following events

- $\mathcal{E}_1 = \big(r_u = \mathtt{dist}_G(v, u)\big)$.
- $\mathcal{E}_2 = \big(r_u \geqslant \mathtt{dist}_G(v, u)\big)$.
- $\mathcal{E}_3 = \big(\forall u' \in \mathtt{before}(u), r_{u'} < \mathtt{dist}_G(v, u')\big)$.

We have

$$\begin{aligned} &\Pr[v \textit{ is not clustered } \mid \mathtt{Center}(v) = u] \\ = \;&\Pr[\mathcal{E}_1 \cap \mathcal{E}_3 \mid \mathcal{E}_2 \cap \mathcal{E}_3] \\ = \;&\frac{\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3]}{\Pr[\mathcal{E}_2 \cap \mathcal{E}_3]} \\ = \;&\frac{\Pr[\mathcal{E}_1 \cap \mathcal{E}_3]}{\Pr[\mathcal{E}_2 \cap \mathcal{E}_3]} \\ = \;&\frac{\Pr[\mathcal{E}_3] \cdot \Pr[\mathcal{E}_1 | \mathcal{E}_3]}{\Pr[\mathcal{E}_3] \cdot \Pr[\mathcal{E}_2 | \mathcal{E}_3]} \\ = \;&\frac{\Pr[\mathcal{E}_1]}{\Pr[\mathcal{E}_2]} = \varepsilon, \end{aligned}$$

where in the penultimate equality, we used the property that the event $\mathcal{E}_3$ is independent of events $\mathcal{E}_1$ and $\mathcal{E}_2$, and the last equality follows from the probability distribution function of the exponential distribution (recall that this is exactly the *memoryless property* of the exponential distribution). Hence, we can now go back and say that

$$\begin{aligned} \Pr & [v \textit{ is not clustered }] \\ = & \sum_{u \in V} \Pr[v \textit{ is not clustered } \mid \mathtt{Center}(v) = u] \cdot \Pr[\mathtt{Center}(v) = u] \\ = & \sum_{u \in V} \varepsilon \cdot \Pr[\mathtt{Center}(v) = u] = \varepsilon. \end{aligned}$$

□

Corollary 1.34. *After $\mathcal{C} = O(\log_{1/\varepsilon} n)$ iterations, all nodes are clustered, with high probability.*

1.5.3 Deterministic Construction I — Subpolynomial Time

In this section, we discuss the classic deterministic algorithms for network decomposition, from 1990s, which achieves the following statement.

Theorem 1.35. *There is a deterministic* LOCAL *algorithm that computes a $(\mathcal{C}, \mathcal{D})$ strong-diameter network decomposition of any n-node graph G, for $\mathcal{C} = 2^{O(\sqrt{\log n})}$ and $\mathcal{D} = 2^{O(\sqrt{\log n})}$, in $2^{O(\sqrt{\log n})}$ rounds.*

In the exercises, we will see how to use this algorithm to compute an $(O(\log n), O(\log n))$ strong-diameter network decomposition, still in the same running time of $2^{O(\sqrt{\log n})}$. The result of Theorem 1.35 is due to [PS92]. Here, we will present a slightly weaker but simple result, due to [ALGP89], that provides the following slightly weaker bounds:

Theorem 1.36. *There is a deterministic* LOCAL *algorithm that computes a $(\mathcal{C}, \mathcal{D})$ strong-diameter network decomposition of any n-node graph G, for $\mathcal{C} = 2^{O(\sqrt{\log n \log\log n})}$ and $\mathcal{D} = 2^{O(\sqrt{\log n \log\log n})}$, in $2^{O(\sqrt{\log n \log\log n})}$ rounds.*

Towards this goal, we first need to introduce a helper tool, *ruling sets*, and present an efficient algorithm for computing them.

Ruling Sets

Definition 1.37. *Given a graph $G = (V, E)$ and a set $W \subseteq V$, an (α, β)-ruling set of W in G is a subset $S \subseteq W$ such that the following two properties are satisfied:*

(A) For each two vertices $v, u \in S$, we have $\mathtt{dist}_G(v, u) \geqslant \alpha$

(B) *For each vertex $v \in W$, there exists a vertex $u \in S$ such that* $\mathrm{dist}_G(v,u) \leqslant \beta$.

For instance, if $W = V$, a maximal independent set $S \subset V$ is simply a $(2,1)$-ruling set. Moreover, letting G^k be the supergraph of G where each two vertices with distance at most k are connected, a set $S \subset V$ that is a maximal independent set in G^k is actually a $(k+1,k)$-ruling set in G.

Lemma 1.38. *Given a graph $G = (V,E)$ and a set $W \subseteq V$, there is a deterministic* LOCAL *algorithm that computes a $(k, k\log n)$-ruling set of W in G, in $O(k\log n)$ rounds.*

Proof. The algorithm is recursive. Let W_0 be the set of vertices whose identifier ends in a 0 bit, and let W_1 be the set of vertices whose identifier ends in a 1 bit. Recursively compute $(k, k(\log n - 1))$-ruling sets S_0 and S_1 of W_0 and W_1, respectively, in $O(k(\log n - 1))$ rounds. Notice that the parameter of the recursion is the length of the binary representation of the identifiers. Now, let $S = S_0 \cup S_1^*$ where S_1^* is the set of all vertices in S_1 who do not have any S_0-vertex within their distance k. Note that S can be computed from S_0 and S_1, in k rounds. One can see that S is a (k,β)-ruling set of W for $\beta = k(\log n - 1) + k = k\log n$ (why?). □

Constructing the Network Decomposition

Here, we describe the network decomposition algorithm that establishes Theorem 1.36. The construction uses a free parameter d, which we will set later on, in order to optimize some trade off.

The construction is iterative, and works in $\log_d n$ similar iterations. Let us start with the description of the first iteration.

Partition vertices into two classes, high-degree vertices H whose degree is at least d, and low-degree vertices L whose degree is at most $d-1$. Compute a $(3, O(\log n))$-ruling set S of the H vertices in G, in $O(\log n)$ rounds, using Lemma 1.38. Now for the set of all vertices in V who have at least one S-vertex within their distance $O(\log n)$, bundle them around the closest S-vertex, breaking ties based on the identifiers. Hence, each bundle induces a subgraph of radius at most $O(\log n)$ and has at least $d+1$ vertices (why?). Furthermore, the set of nodes that remain unbundled induces a graph with maximum degree at most $d-1$.

Compute a d-coloring of the subgraph induced by unbundled vertices in $O(d + \log^* n)$ rounds, using the algorithm we discussed in Theorem 1.25. Each of these d colors will be one of the subgraphs G_i in our network decomposition's partition, and each vertex of each color class is simply its own cluster. Note that clearly the clusters of the same class are non-adjacent.

We are now essentially done with the first iteration. To start the second iteration, we will switch to a new graph $\mathcal{G}_2$, defined as follows. We will imagine contracting each bundle into a single new node in $\mathcal{G}_2$. We call these the level-2 nodes. In $\mathcal{G}_2$, two level-2 nodes are connected if their bundles include adjacent vertices in $G = \mathcal{G}_1$. Notice that one round of communication on $\mathcal{G}_2$ can be performed in $O(\log n)$ rounds of communication on the base graph $G = \mathcal{G}_1$, simply because each of the bundles has diameter $O(\log n)$.

Now the construction for level-2 works similar to that of level-1, but on the graph $\mathcal{G}_2$. Again, we partition level-2 vertices into high and low-degree, by comparing to threshold d. Then, we compute a $(3, O(\log n))$-ruling set of high-degree level-2 vertices, with respect to the graph $\mathcal{G}_2$. We then bundle nodes around these ruling set vertices into bundles of diameter $O(\log n)$ in $\mathcal{G}_2$. Afterward, we color the level-2 vertices that remain unbundled using d colors. Notice that each level-2 vertex is actually one of the bundles in level-1, and hence, by this coloring, we color all the vertices of that bundle using the same color. Therefore, this is not a proper color of $\mathcal{G}_1$, but each color of this level does induce a block satisfying the conditions of network decomposition (why?).

Afterward, we repeat to level 3, by contracting the bundles of level-2, producing the graph $\mathcal{G}_3$ as a result, and continuing the process on $\mathcal{G}_3$.

Lemma 1.39. *Each level i bundle has at least $(d+1)^{i-1}$ vertices. Therefore, $\log_d n$ levels of iteration suffice.*

Proof. Follows by a simple induction and observing that each level i bundle includes at least $d+1$ level $(i-1)$ bundles. □

Lemma 1.40. *Each level i bundle has diameter $O(\log n)^i$ in G. Hence, the algorithm for level i can be performed in $O(\log n)^i \cdot O(d + \log^* n)$ rounds.*

Proof. Follows by a simple induction and observing that each level i bundle is a graph of diameter $O(\log n)$ on the level $(i-1)$ graph $\mathcal{G}_{i-1}$. □

Corollary 1.41. *Overall the algorithm takes at most $d \cdot O(\log n)^{\log_d n}$ rounds. At the end of all levels, we get a $(\mathcal{C}, \mathcal{D})$ strong-diameter network decomposition for $\mathcal{C} = d \log_d n$ and $\mathcal{D} = O(\log n)^{\log_d n}$. Setting $d = 2^{O(\sqrt{\log n \log\log n})}$, we get round complexity of $d \cdot O(\log n)^{\log_d n} = 2^{O(\sqrt{\log n \log\log n})}$ and $\mathcal{C} = d \log_d n = 2^{O(\sqrt{\log n \log\log n})}$ and $\mathcal{D} = O(\log n)^{\log_d n} = 2^{O(\sqrt{\log n \log\log n})}$.*

1.5.4 Deterministic Construction II — Polylogarithmic Time

In this section, we discuss a recent algorithm of [RG20] that computes a network decomposition with ideal parameters in poly$(\log n)$ rounds[9]. This

[9]We note that the writing here is borrowed almost verbatim from [RG20], modulo some editorial aspects.

leads to the first $\text{poly}(\log n)$ rounds deterministic algorithms for a wide range of problems, including $\Delta + 1$ coloring, maximal independent set, etc.

Concretely, the statement of the network decomposition result is as follows:

Theorem 1.42. *There is a deterministic* LOCAL *algorithm that computes a* $(\mathcal{C}, \mathcal{D})$ *strong-diameter network decomposition of any* n*-node graph* G*, for* $\mathcal{C} = O(\log n)$ *and* $\mathcal{D} = O(\log n)$*, in* $O(\log^8 n)$ *rounds.*

To focus on the core algorithmic part of the construction, we present the proof for a seemingly weaker statement, as provided below in Theorem 1.43. In the exercises, we will see how one can use the algorithm of Theorem 1.43 in a black-box manner to achieve Theorem 1.42. Moreover, for some of the main applications (e.g., deterministic algorithms for coloring, MIS, etc), Theorem 1.43 itself is directly sufficient.

Theorem 1.43. *There is a deterministic* LOCAL *algorithm that computes a* $(\mathcal{C}, \mathcal{D})$ *weak-diameter network decomposition of any* n*-node graph* G*, for* $\mathcal{C} = O(\log n)$ *and* $\mathcal{D} = O(\log^3 n)$*, in* $O(\log^7 n)$ *rounds.*

We obtain Theorem 1.43 by $c = \log n$ iterations of applying the following clustering lemma. Intuitively, each iteration of applying this lemma gives the clusters in one color of the network decomposition. More concretely, this lemma's algorithm clusters at least half of the vertices—more precisely, half of the vertices that remain unclustered—into non-adjacent clusters, each with weak-diameter at most $O(\log^3 n)$. The precise statement is as follows:

Lemma 1.44. *(**Clustering Lemma**) Consider an arbitrary* n*-node network graph* $G = (V, E)$ *where each node has a unique* $b = O(\log n)$*-bit identifier, as well as a set* $S \subseteq V$ *of living vertices. There is a deterministic distributed algorithm that, in* $O(\log^6 n)$ *rounds in the* LOCAL *model, finds a subset* $S' \subseteq S$ *of living vertices, where* $|S'| \geqslant |S|/2$*, such that the subgraph* $G[S']$ *induced by set* S' *is partitioned into non-adjacent disjoint clusters, each of weak-diameter* $O(\log^3 n)$ *in graph* G*.*

We obtain Theorem 1.43 directly by $c = \log n$ iterations of applying Lemma 1.44, starting from $S = V$. For each iteration $j \in [1, \log n]$, the set S' are exactly nodes of color j in the network decomposition, and we then continue to the next iteration by setting $S \leftarrow S \setminus S'$. In the remaining part of this section, we focus on presenting the algorithm of Lemma 1.44 and its analysis.

The Clustering Lemma's Algorithm

We now describe the algorithm outline of Lemma 1.44. The construction has $b = O(\log n)$ phases, corresponding to the number of bits in the identifiers.

Initially, we think of all nodes of S as *living*. During this construction, some living nodes *die*. We use S'_i to denote the set of living vertices at the beginning of phase $i \in [0, b-1]$. Slightly abusing the notation, we let S'_b denote the set of living vertices at the end of phase $b-1$ and define S' to be the final set of living nodes, i.e., $S' := S'_b$.

Moreover, we *label* each living node v with a b-bit string $\ell(v)$, and we use these labels to define the clusters. At the beginning of the first phase, $\ell(v)$ is simply the unique identifier of node v. This label can change over time. For each b-bit label $L \in \{0,1\}^b$, we define the corresponding *cluster* $S'_i(L) \subseteq S'_i$ in phase i to be the set of all living vertices $v \in S'_i$ such that $\ell(v) = L$.

Construction Invariants The construction is such that, for each phase $i \in [0, b-1]$, we maintain the following invariants:

(I) For each i-bit string Y, the set $S'_i(Y) \subseteq S'_i$ of all living nodes whose label ends in suffix Y has no edge to other living nodes $S'_i \setminus S'_i(Y)$. In other words, the set $S'_i(Y)$ is a union of some connected components of the subgraph $G[S'_i]$ induced by living nodes S'_i.

(II) For each label L, the corresponding cluster $S'_i(L)$ has weak-radius at most iR, where $R = O(\log^2 n)$. We emphasize that in the subgraph induced by living vertices a cluster can be disconnected.

(III) We have $|S'_{i+1}| \geqslant |S'_i|(1 - 1/2b)$.

These invariants prove Lemma 1.44: Invariant (I) shows that at the end of b phases, different clusters are non-adjacent. Invariant (II) shows that each cluster has weak-radius $bR = O(\log^3 n)$. Invariant (III) shows that for the final set of living nodes $S' = S'_b$, we have $|S'| \geqslant (1 - 1/2b)^b |S| \geqslant |S|/2$.

Outline of One Phase We now outline the construction of one phase and describe its goal. Let us think about some fixed phase i. We focus on one specific i-bit suffix Y and the respective set $S'_i(Y)$. Let us categorize the nodes in $S'_i(Y)$ into two groups of *blue* and *red*, based on whether the $(i+1)^{\text{th}}$ least significant bit of their label is 0 or 1. Hence, all blue nodes have labels of the form $(* \ldots * 0Y)$ and all red nodes have labels of the form $(* \ldots * 1Y)$, where $*$ can be an arbitrary bit. Please also see Figure 1.1, when following these descriptions. During this phase, we make some small number of the red vertices die and we change the labels of some of the other red vertices to blue labels (and then the node is also colored blue). All blue nodes remain living and keep their label. The eventual goal is that, at the end of the phase, among the living nodes, there is no edge from a blue node to a red node. Hence, each connected component of the living nodes consists either entirely of blue nodes or entirely of red nodes. Therefore, the length of the common suffix in each connected component is incremented, which leads to invariant (I) for the next phase. The construction ensures that we kill at most $|S'_i(Y)|/2b$ red vertices of set $S'_i(Y)$, during this phase, which gives invariant (III). We next describe the steps of this phase.

Steps of One Phase Each phase consists of $R = 10b \log n = O(\log^2 n)$ steps, each of which will be implemented in $O(\log^3 n)$ rounds. Hence, the overall round complexity of one phase is $O(\log^5 n)$ and over all the $O(\log n)$ phases, the round complexity of the whole construction of Lemma 1.44 is $O(\log^6 n)$ as advertised in its statement. Each step of the phase works as follows: each red node sends a request to an arbitrary neighboring blue cluster, if there is one, to join that blue cluster (by adopting the label). For each blue cluster A, we have two possibilities:

(1) If the number of adjacent red nodes that requested to join A is less than or equal to $|A|/2b$, then A does not accept any of them and all these requesting red nodes die (because of their request being denied by A). In that case, cluster A *stops* for this whole phase and does not participate in any of the remaining steps of this phase.

(2) Otherwise — i.e., if the number of adjacent red nodes that requested to join A is strictly greater than $|A|/2b$ — then A accepts all these requests and each of these red nodes change their label to the blue label that is common among all nodes of A. In this case, the cluster A grows by at most one hop to include all these newly joined nodes.

We note that each step can be performed in $O(\log^3 n)$ rounds, because each blue cluster has radius at most $O(\log^3 n)$ and therefore can gather the number of vertices in the cluster, as well as the number of red vertices that would like to join this cluster. We also emphasize that in each step, each red node acts alone, independent of other nodes in the same red cluster. Hence, red clusters may shrink, disconnect, or even get dissolved over time. Once a red node adopts a blue label (or if a node had a blue label at the beginning), it will maintain that label throughout the phase. Therefore, blue clusters can only grow, and have more and more red nodes join them. We also emphasize that we can have blue clusters adjacent to each other, and red clusters adjacent to each other – the objective is to have no edge connecting a red cluster to a blue cluster.

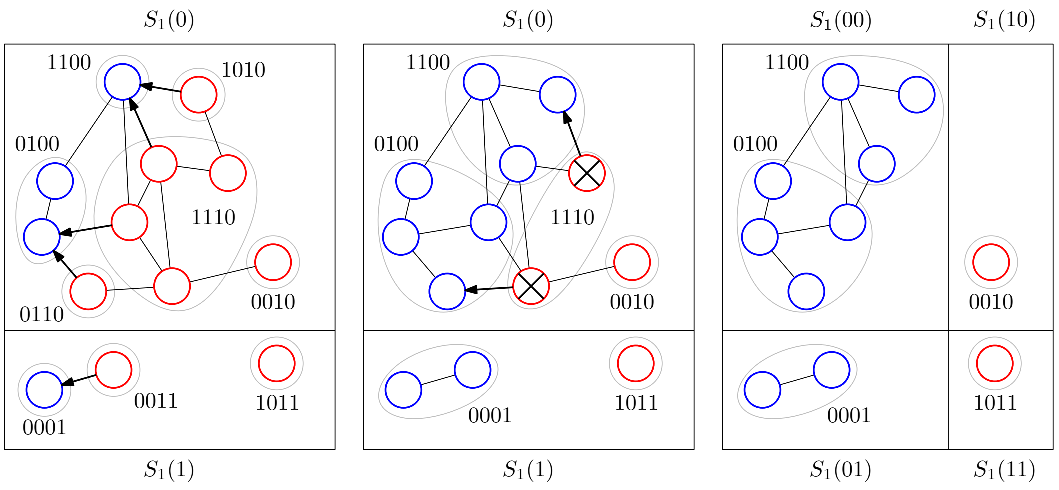

Figure 1.1: In this illustration, we consider the second phase of the algorithm, in a simple example graph. The three figures show the configuration in the beginning of three steps of this phase from left to right. Note that, at the beginning of this phase, the clusters are already separated according to their least significant bit (as a result of the first phase). When the second phase starts—i.e., in the left figure—the second least significant bit determines whether each cluster is blue or red. Adjacent red nodes are proposing to blue nodes (dark arrows) to join their clusters and blue clusters decide either to relabel them so that they join this cluster or to make them die (crossed red vertices). In the end, blue and red clusters are separated. Note that nothing will happen in the third phase, since the only two adjacent clusters share the same bit on the third least significant bit. Their boundary will be resolved only in the last phase.

Analysis of the Clustering Algorithm

We next provide some simple observations about this construction in one phase, which allow us to argue that the construction maintains invariants (I) to (III), described above.

Observation 1.45. *Any blue cluster stops after at most $4b \log n$ steps.*

Proof. In each step that a cluster A does not stop, its size grows by a factor of at least $(1 + 1/2b)$, as it accepts at least $|A|/2b$ requests from neighboring red nodes. Hence, after $4b \log n$ steps of growth, the size would exceed $(1 + 1/2b)^{4b \log n} > n$, which is not possible. Therefore, cluster A stops after at most $4b \log n$ steps. □

Observation 1.46. *Once a blue cluster A stops, it has no edge to a red node, and it will never have one during this phase. This implies invariant (I).*

Proof. By the observation above, cluster A stops after at most $4b \log n$ steps. Consider the step in which cluster A stops. In that step, each neighboring red node (if there is one) either requested to join A or some other blue cluster. In the former case, that red node dies. In the latter case, the node adopts a blue label or dies. In either case, the node is not a living red node anymore (and it will never become one). From this point onward, this blue cluster A never grows or shrinks. □

Observation 1.47. *In each step, the radius of each blue cluster grows by at most* 1*, while the radius of each red cluster does not grow. This implies invariant (II).*

Observation 1.48. *The total number of red vertices in* $S'_i(Y)$ *that die during this phase is at most* $|S'_i(Y)|/(2b)$*. This implies invariant (III).*

Proof. From Observation 1.46, it follows that each blue cluster A stops exactly once, and if it had $|A|$ vertices at that point, it makes at most $|A|/(2b)$ red vertices die. Hence, in total over the whole phase, the number of red vertices that die is at most a $1/(2b)$ fraction of the number of nodes in blue clusters that stop, and thus at most $|S'_i(Y)|/(2b)$. □

These observations show that the constructions satisfies invariants (I) to (III). As discussed when introducing the invariants, this completes the proof of Lemma 1.44.

1.6 Maximal Independent Set

The Maximal Independent Set (MIS) problem is a central problem in the area of distributed algorithms for local graph problems. One partial reason for this central role is that many other problems, including graph coloring and maximal matching, reduce to MIS, as we soon see.

1.6.1 Definition and Reductions

Let us start with recalling the definition of MIS:

Definition 1.49. *Given a graph* $G = (V, E)$*, a set of vertices* $S \subseteq V$ *is called a* Maximal Independent Set (MIS) *if it satisfies the following two properties:*

(1) the set S *is an* independent *set meaning that no two vertices* $v, u \in S$ *are adjacent,*

(2) the set S *is* maximal *— with regard to independence — meaning that we cannot add any node* $v \notin S$ *to the set* S*, i.e., there exists a neighbor* u *of* v *such that* $u \in S$*.*

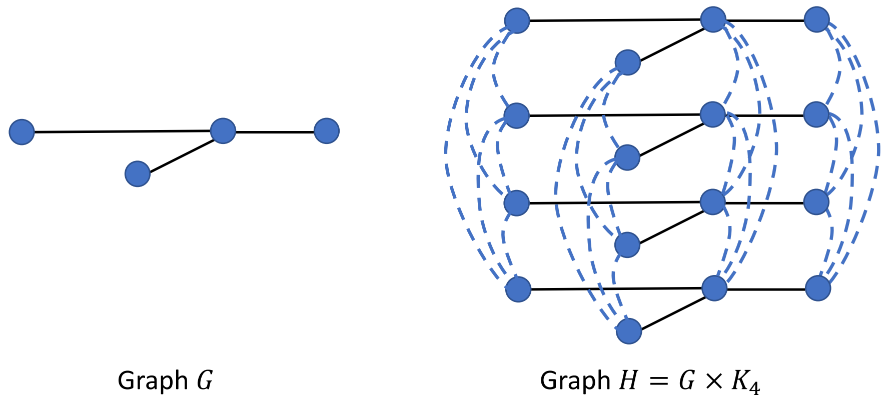

Figure 1.2: A simple graph G and its transformed version $H = G \times K_{\Delta+1}$.

The *independence* condition ensures that we choose nodes that are not neighboring each other. For instance, this is what we desire when we want neighboring nodes to avoid performing their operations simultaneously, e.g., when accessing a shared resource, such as a common channel in wireless networks or the same memory location in multi-processor computers. The maximality condition then ensures that we cannot add any more nodes to the solution, without destroying the independence property. In some sense, the nodes in an MIS can be seen as centers for clustering: each node will be either itself such a center or will have a neighbor as a center, while the centers are not adjacent.

More concretely, besides the above intuitive description of applications, algorithms for MIS can be used to solve a number of other graph problems. Below, we see a simple and beautiful reduction that allows us to solve $\Delta+1$ coloring using an MIS algorithm, without any significant overhead in the round complexity.

Lemma 1.50. *Given a* LOCAL *algorithm $\mathcal{A}$ that computes an MIS on any N-node graph in $T(N)$ rounds, there is a* LOCAL *algorithm $\mathcal{B}$ that computes a $\Delta+1$ coloring of any n-node graph with maximum degree Δ in $T(n(\Delta+1))$ rounds.*

In particular, we will soon see an $O(\log n)$ round randomized algorithm for computing an MIS on n-node graphs, which by this lemma, implies an $O(\log n)$ round randomized algorithm for $(\Delta+1)$ coloring.

Proof of Lemma 1.50. Let G be an arbitrary n-node graph with maximum degree Δ, for which we would like to compute a $(\Delta+1)$-coloring.

Let $H = G \times K_{\Delta+1}$ be an $n(\Delta+1)$-vertex graph generated by taking $\Delta+1$ copies of G. See Figure 1.2 for an example illustration. Hence each node $v \in G$ has $\Delta+1$ copies in H, which we will refer to as $v_1, v_2, \ldots, v_{\Delta+1}$. Then, add additional edges between all copies of each node $v \in G$, that is, each two copy vertices v_i and v_j are connected in H.

Run the algorithm $\mathcal{A}$ on H. The resulting MIS produces a maximal independent set S. For each node $v \in G$, the color of v will be the number i such that the $v_i \in S$. Clearly, each node receives at most one color, as at most one copy v_i of $v \in G$ can be in S. However, one can see that each node $v \in G$ receives actually exactly one color. The reason is that each neighboring node $u \in G$ can have at most one of its copies in S. Node v has at most Δ neighbors u and $\Delta + 1$ copies v_i. Hence, there is at least one copy v_i of v for which no adjacent copy u_i of neighboring vertices $u \in G$ is in the set S. By maximality of S, we must have $v_i \in S$. □

1.6.2 Luby's MIS Algorithm

As we see next, there is an extremely simple and elegant randomized algorithm that computes an MIS in merely $O(\log n)$ rounds, with high probability (probability at least $1 - 1/n$). This is a celebrated result of Luby [Lub85] and Alon, Babai, and Itai [ABI86] from the 1980s, for which they received the 2016 Dijkstra award.

Luby's Algorithm: The algorithm is made of iterations, each of which has two rounds, as follows:

- In the first round, each node v picks a random real number $r_v \in [0, 1]$ and sends it to its neighbors[10]. Then, node v joins the (eventual) MIS set S if and only if node v has a strict local maxima, that is, if $r_v > r_u$ for all neighbors u of v.

- In the second round, if a node v joined the MIS, then it informs its neighbors and then, node v and all of its neighbors get removed from the problem. That is, they will not participate in the future iterations.

Analysis: It is easy to see that the algorithm always produces an independent set, and eventually, this set is maximal. The main question is, how long does it take for the algorithm to reach a maximal independent set?

Theorem 1.51. *Luby's Algorithm computes a maximal independent set in $O(\log n)$ rounds, with high probability.*

Proof. Let us first give a brief proof outline. Consider an arbitrary iteration i and suppose that the graph induced on the remaining vertices is G_i, which

[10]One can easily see that having real numbers is unnecessary and values with $O(\log n)$-bit precision suffice.

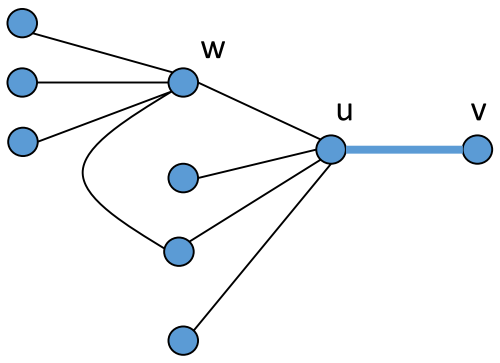

Figure 1.3: Node w killing edge $e = \{u, v\}$

has m_i edges. In the following, and as the main ingredient of the proof, we show that the graph G_{i+1} which will remain for the next iteration has in expectation at most $m_i/2$ edges. Then, at the end, we use a repeated application of this to conclude that after $4 \log n$ iterations, the leftover graph is empty with high probability and thus the algorithm terminates.

As outlined above, the main step is to consider the graph G_i remaining for step i, and to show that

$$\mathbb{E}[m_{i+1} \mid G_i \textit{ is any graph with } m_i \textit{ edges}] \leqslant m_i/2.$$

In the above, m_{i+1} denotes the number of edges in the graph that remains at the end of iteration i. To prove this inequality, let us consider an edge $e = \{u, v\}$ and a neighbor w of u, as depicted in Figure 1.3. If w has the maximum number among $N(w)$, the set of neighbors of w in the remaining graph, then w joins the MIS and hence nodes w and u and thus also the edge $e = \{u, v\}$ get removed. In this case, we say node w killed edge $e = \{u, v\}$. The probability of w having the maximum number among its neighbors is exactly $1/(d(w) + 1)$, where $d(w)$ denotes the degree of node w. But, we are interested in the probability of edge e being killed by any such neighbor w. We cannot easily sum over the probabilities of the neighbors w_1, w_2, ..., killing edge e, as those events are not necessarily disjoint — several of them might happen at the same time, in which case summing the probabilities would be over counting.

To circumvent this, we make a slight adjustment: we say that node w single-handedly kills $e = \{u, v\}$ (from the side of u) if r_w is the maximum random number among those of nodes in $N(w) \cup N(u)$. Notice that this limits the number of double-counting of an edge being killed to 2, meaning that at most one node w might single-handedly kill $e = \{u, v\}$ (from the side of u) and at most one node w' might single-handedly kill $e = \{u, v\}$ (from the side of v). Hence, we can lower bound the number of removed edges by bounding the number of single-handedly killed edges and dividing that by 2. More concretely, suppose we use the indicator random variable $X_{w\to e}$ which is equal to 1 if and only if node $w \in N(e)$— where $N(e) = N(u) \cup N(v)$ is the set of all nodes that are adjacent to at least one endpoint of e—single-handedly kills edge $e = \{u, v\}$. If the edge e is removed, then $(\sum_{w\in N(e)} X_{w\to e})/2$ is at least 1/2 and at most 1. Hence, $(\sum_{w\in N(e)} X_{w\to e})/2$ is a (good) lower bound for the indicator of whether edge e is removed or not. Thus, we can lower bound the expected number of removed edges over the entire graph G_i as follows:

$$\mathbb{E}[m_i - m_{i+1}] \geqslant \mathbb{E}\Bigg[\sum_{e\in E}(\sum_{w\in N(e)} X_{w\to e})/2\Bigg] = \sum_{e\in E}\sum_{w\in N(e)} \mathbb{E}[X_{w\to e}]/2.$$

To analyze this sum, we rearrange the summations and put the one on nodes w on the outside:

$$\sum_{e\in E}\sum_{w\in N(e)} \mathbb{E}[X_{w\to e}]/2 = \sum_{w\in V}\sum_{e\in E \text{ s.t. } w\in N(e)} \mathbb{E}[X_{w\to e}]/2.$$

Now, we can focus on the contributions of each node w to this summation (grouping the edges e by their other endpoint u), as follows: Consider two neighboring nodes w and u. The probability that w has the maximum among $N(w) \cup N(u)$ is at least $\frac{1}{d(w)+d(u)}$. In that case, node w single-handedly kills $d(u)$ edges incident on u (from the side of u). Similarly, the probability that u has the maximum among $N(w) \cup N(u)$ is at least $\frac{1}{d(w)+d(u)}$, and in that case, u single-handedly kills $d(w)$ edges incident on w (from the side of w). Therefore, we can lower bound the total expected number of removed edges as follows:

$$\begin{aligned}
\mathbb{E}[m_i - m_{i+1}] &\geqslant \sum_{w\in V}\sum_{e\in E \text{ s.t. } w\in N(e)} \mathbb{E}[X_{w\to e}]/2 \\
&= \sum_{\{w,u\}\in E}\Bigg(\sum_{e\in E \text{ s.t. } u\in N(e)} \mathbb{E}[X_{w\to e}]/2 + \sum_{e\in E \text{ s.t. } w\in N(e)} \mathbb{E}[X_{u\to e}]/2\Bigg) \\
&\geqslant \sum_{\{w,u\}\in E_i}(\frac{d(u)}{d(w)+d(u)} + \frac{d(w)}{d(w)+d(u)})/2 = m_i/2.
\end{aligned}$$

Now we know that we have $\mathbb{E}[m_{i+1} \mid G_i$ *is any graph with* m_i *edges*$] \leqslant m_i/2$, for every iteration i. By a repeated application of this inequality over different iterations, we get that the expected number of the edges in the graph that remains after $4\log n$ iterations is $\mathbb{E}[m_{4\log n}] \leqslant m_0/2^{4\log n} \leqslant 1/n^2$ edges. Hence, by Markov's inequality, the probability that the graph $G_{4\log n}$

has at least 1 edge is at most $1/n^2$. That is, with high probability, $G_{4\log n}$ has no edge left, and thus, the algorithm finishes in $4\log n + 1$ iterations, with high probability. $\square$

1.7 Sublogarithmic-Time Randomized Coloring

Here, we explain a randomized algorithm that in $O(\sqrt{\log n})$ rounds, achieves a coloring with almost $\Delta + 1$ colors:

Theorem 1.52. *There is a randomized* LOCAL *algorithm that computes a $\Delta(1+\varepsilon)$-coloring in $O(\sqrt{\log n})$ rounds*[11]*, for any constant $\varepsilon > 0$, with high probability.*

We present the algorithm in two parts: First, in Section 1.7.1, we explain the algorithm assuming that $\Delta \leqslant 2^{\sqrt{\log n}}$, and then, in Section 1.7.2, we extend the algorithm to larger Δ using a small additional step.

1.7.1 The algorithm for low-degree graphs

Theorem 1.53. *There is a randomized algorithm that computes a $\Delta(1+\varepsilon)$ coloring of the graph in $\Theta(\sqrt{\log n}/\varepsilon)$ rounds, for any constant $\varepsilon > 0$, assuming*[12] *that $\Delta \leqslant 2^{\sqrt{\log n}}$.*

Suppose that each vertex has degree at most $2^{\sqrt{\log n}}$. Consider running the following algorithm for $\Theta(\sqrt{\log n}/\varepsilon)$ iterations. In each iteration, each node v picks a random color among the colors not previously taken by any of its neighbors. If no neighbor of v picked the same color, then v takes this color as its permanent coloring, and gets removed from the problem. The neighbors update their palette of remaining colors, by removing the colors taken by the colored neighbors.

In Lemma 1.54 below, we show that after $\Theta(\sqrt{\log n}/\varepsilon)$ iterations for any $\varepsilon \in (0, 1]$, with high probability, each connected component of the remaining graph has diameter at most $\Theta(\sqrt{\log n}/\varepsilon)$. Hence, each of these components can be colored in $\Theta(\sqrt{\log n}/\varepsilon)$ additional steps, deterministically. Therefore, once we prove Lemma 1.54, we have essentially completed the proof of Theorem 1.53.

Lemma 1.54. *After $\Theta(\sqrt{\log n}/\varepsilon)$ iterations, with high probability, each connected component in the subgraph induced by the remaining nodes has at diameter at most $\Theta(\sqrt{\log n}/\varepsilon)$.*

[11]We remark that a much faster algorithm, with a round complexity of $2^{O(\sqrt{\log\log n})}$, is known. We will cover that result in the next sections.

[12]As mentioned before, we will soon see how to remove this assumption.

Proof. We show that with high probability, no path of length $C\sqrt{\log n/\varepsilon}$ can have all of its vertices remain, for a large enough constant $C > 3$.

We start with some simple observations. Notice that in every iteration, the palette size of each node is at least a $(1+\varepsilon)$ factor larger than its number of remaining neighbors, i.e., its degree in the remaining graph. Hence, in each iteration, each node gets colored with probability at least $\varepsilon/(1+\varepsilon) \geqslant \varepsilon/2$, even independent of its neighbors (why?).

Consider an arbitrary path $P = v_0, v_1, \dots, v_\ell$ where $\ell = \Theta(\sqrt{\log n}/\varepsilon)$. In each iteration, each of these vertices gets removed with probability at least ε, regardless of the coloring of the other vertices. Hence, the probability that all these nodes remain for $k = C\sqrt{\log n}/\varepsilon$ iterations is at most

$$(1-\varepsilon/2)^{\ell k} \leqslant \exp(-\ell k \varepsilon/2) \leqslant \exp(-C^2 \log n/(2\varepsilon)).$$

Now, there are at most $n\Delta^\ell$ ways for choosing the path P, because there are n choices for the starting point and then Δ choices for each next hop. Hence, by a union bound, the probability that any such path remains is at most

$$\begin{aligned}
& n \cdot \Delta^\ell \cdot \exp(-C \log n/(2\varepsilon)) \\
= \; & \exp(\log n + \ell \log \Delta - C^2 \log n/(2\varepsilon)) \\
\leqslant \; & \exp(\log n + C\sqrt{\log n/\varepsilon} \cdot \sqrt{\log n} - C^2 \log n/(2\varepsilon)) \\
= \; & \exp(\log n + C \log n/\varepsilon - C^2 \log n/(2\varepsilon)) \leqslant 1/n^C.
\end{aligned}$$

□

1.7.2 The extension to high-degree graphs

We now see how to extend Theorem 1.53 to higher degree graphs, using one extremely simple step.

Theorem 1.55. *There is a randomized algorithm that computes a $\Delta(1+\varepsilon)$ coloring of the graph in $\Theta(\sqrt{\log n}/\varepsilon)$ rounds, for any constant $\varepsilon > 0$.*

Suppose that $\Delta \geqslant 2^{\sqrt{\log n}}$, as otherwise Theorem 1.53 suffices. Partition G into $k = \alpha\varepsilon^2\Delta/\log n$ vertex-disjoint subgraphs G_1, G_2, G_3, …, G_k by putting each vertex in one of these subgraphs at random. Here, α is a sufficiently small positive constant $\alpha > 0$. In Lemma 1.56, we argue that, with high probability, each subgraph has degree at most $\frac{\Delta}{k}(1+\varepsilon/3) = O(\log n)$. This will be by a simple application of the Chernoff bound. Hence, it can be colored using $\frac{\Delta}{k}(1+\varepsilon/3)(1+\varepsilon/3) \leqslant \frac{\Delta}{k}(1+\varepsilon)$ colors, via the method of Theorem 1.53, in $\Theta(\sqrt{\log n}/\varepsilon)$ rounds. We use different colors for different subgraphs, and color them all in parallel. Hence, overall, we get a coloring with $k \cdot \frac{\Delta}{k}(1+\varepsilon) = \Delta(1+\varepsilon)$ colors, in $\Theta(\sqrt{\log n}/\varepsilon)$ rounds.

Lemma 1.56. *With high probability, each subgraph* G_i *has maximum degree at most* $\frac{\Delta}{k}(1+\varepsilon/3)$.

The lemma follows in a straightforward manner from the Chernoff bound, and a union bound. The Chernoff bound is among the most basic concentration of measure tools. In a rough sense, it shows that the sum of independent (indicator) random variables has a distribution well-concentrated around its expected value. More concretely, the probability that this sum deviates significantly from its expected value is exponentially small in the expected value. The mathematical statement is as follows:

Theorem 1.57. *(Chernoff Bound) Suppose* $X_1, X_2, \dots, X_\eta$ *are independent random variables taking values in* $\{0, 1\}$. *Let* $X = \sum_{i=1}^{\ell} X_i$ *denote their sum and let* $\mu = \mathbb{E}[X]$ *denote the sum's expected value. For any* $0 < \delta \leqslant 1$, *we have*

$$\Pr[X \notin [\mu(1-\delta), \mu(1+\delta)] \leqslant 2e^{-\delta^2\mu/3}.$$

Proof of Lemma 1.56. Consider each node $v \in G$. Let X_1, X_2, $\dots X_\Delta$ be the indicator random variables of whether the i^{th} neighbor of v picks the same subgraph as v does. Let $X = \sum_{i=1}^{\ell} X_i$ and $\mu = \mathbb{E}[X]$. Notice that $\mu = \Delta/k$. Given that $k = \alpha\varepsilon^2\Delta/\log n$, we get that $\mu \geqslant \frac{\log n}{\alpha\varepsilon^2}$. Hence, by Chernoff bound, we have

$$\Pr[X \geqslant \mu(1+\varepsilon/3)] \leqslant 2e^{-\varepsilon^2\mu/18} \leqslant e^{1-\log n/(18\alpha)} \leqslant 1/n^3,$$

where the last inequality holds for small enough α, e.g., $\alpha = 0.01$.

Now, we know that the probability of one node v having a degree (in its own subgraph) higher than the desired threshold $\frac{\Delta(1+\varepsilon/3)}{k}$ is at most $1/n^3$. By a union bound over all nodes v, we get that the probability of having such a node is at most $1/n^2$. In other words, with high probability, each node has degree at most $\frac{\Delta(1+\varepsilon/3)}{k}$ in its own subgraph. □

1.8 Sublinear-Time Centralized Algorithms

In this section, we discuss *sublinear-time centralized algorithms* for graph problems, and particularly what is known as *Local Computation Algorithms* (LCA). As we shall see soon, the core part of these algorithms is a local procedure, quite similar to the LOCAL distributed algorithms that we discussed in the previous sections of this chapter.

Sublinear-time (centralized) algorithms are gaining importance with the constant increase in the size of the graph problems that need to be solved[13]. In particular, for a range of problems of interest, the graphs have

[13]Does *"Big Data"* ring a bell?

become so large that spending a linear-time to read the whole graph to solve the problem is well-beyond the time that we can afford. We instead want centralized algorithms that, on an n-node graph, spend $\Theta(1)$ time or at most $\text{poly}(\log n)$ time and provide some meaningful answer (e.g. approximation) about the problem at hand. As we will see, the prototypical way of achieving such results is via *local algorithms*. In a very rough sense, we will poke the graph at a few random places, and determine the solution at each of those places by performing a local procedure, which checks only a small neighborhood around there. At the end, we try to infer something about the overall solution by putting together the solutions at these few randomly sampled places.

1.8.1 The LCA Model

We consider a graph $G = (V, E)$ with $n = |V|$ nodes and maximum degree Δ. Moreover, we will assume that the graph has no isolated vertex. For the graphs of interest, which are thought to be extremely large and complex, it is typical to assume that degrees are much smaller than the network size. Thus, we will think of Δ as a constant compared to n. The graph is represented by a query-access model, where each query is as follows:

- Query $Q(v, i)$ asks *"who is the* i^{th} *neighbor of node v*?".

We can also access a random node v, chosen uniformly at random from V. The main performance measure is the number of queries, which we will refer to as the algorithm's *query complexity*. The goal would be to have a query complexity which depends only on Δ (aside from some precision and certainty parameters) and not on n. Moreover, we clearly prefer smaller dependencies on Δ, e.g., $\text{poly}(\Delta)$ is preferred to 2^{Δ}.

1.8.2 Approximating Maximum Matching

Consider the problem of computing an approximation of the size of maximum matching. Recall that a matching is a set of edges $M \subseteq E$ such that no two of the edges in M share an end-point. The size of a matching is simply the number of its edges. Computing a maximum matching, or approximating it, are classic optimization problems which have been studied for decades, at the very least since 1965 work of Edmonds [?], which presented a polynomial-time algorithm for computing a maximum matching.

We next discuss an algorithm that computes a $(2 + \epsilon)$-approximation of the size of maximum matching, for a desirably small constant $\epsilon > 0$ — say $\epsilon = 0.01$ — with query-complexity $2^{O(\Delta)}/\epsilon^2$, independent of how large the graph size n is.

The Algorithm's Roadmap We will present a simple way of constructing a *maximal* matching M, namely a random greedy maximal matching procedure, which we will be able to simulate in a local manner. This allow us to pick a set S of randomly sampled nodes and infer whether each sampled node $s \in S$ is matched in the maximal matching or not, using a small number of queries around s. Then, the average fraction of sampled nodes that are matched give us as estimation of the size of the maximal matching M. Since any maximal matching is a 2-approximaiton of the maximum matching, we get an estimator for a 2-approximation of maximum matching. Using a large enough number $k = |S|$ of sampled nodes S, we can adjust the accuracy and certainty of this estimator. We next explain the details of this outline.

1.8.3 A Local Matching Procedure

Maximal Matching A *maximal matching* is a matching $M \subseteq E$ such that we cannot add any edge of $E \setminus M$ to M, without violating the property that we have a matching. Put more positively, in a maximal matching M, we have the property that for each edge $e \in E$, at least one endpoint $v \in e$ is incident on a matching edge $e' \in M$. Notice that a maximal matching is not necessary a maximum matching. However, as we prove next, any maximal matching has a size close to that of a maximum matching.

Lemma 1.58. *In any graph, any maximal matching has size at least* $1/2$ *of the maximum matching.*

Proof. Consider a maximum matching M^* and an arbitrary maximal matching M. We prove that $2|M| \geqslant |M^*|$. For that, we give one dollar to each M^*-edge and then ask this M^*-edge e to pass this dollar to one of its incident edges in M. Each edge has such an incident M-edge due to the maximality of M. This way, each M-edge receives at most two dollars, at most one through each of its endpoints (why?). Hence, we have redistributed $|M^*|$ dollars on the $|M|$ edges of M in a way that each receives at most two dollars. Hence, $2|M| \geqslant |M^*|$. □

Next, we explain a simple algorithm for computing a maximal matching. We will not run this algorithm in its entirety, but rather, we will sample a few nodes and try to locally infer what would be the output of these nodes in the algorithm.

Random Greedy Maximal Matching Algorithm Suppose that for each edge $e \in E$, we pick a random number $r_e \in [0, 1]$. Then, we process the edges in non-decreasing order of their random numbers r_e, and we greedily add them to the matching M. More concretely, as we go through the (non-decreasing) sorted order of the edges, each time we add the edge e under process to the matching M if and only if no edge e' incident to e with a lower random number $r_{e'} < r_e$ has been added to the matching M before. This process always keeps M as a matching, and eventually, once we are done with the processing, M is a maximal matching.

Approximating the Size of Maximal Matching What we would do is as follows. We will pick a set S of k randomly chosen nodes (with replacement). The fraction of these nodes that are matched in M is an unbiased estimator of the fraction of vertices that are matched in M. More precisely,

$$\mathbb{E}[\sum_{s \in S} 1_{(\textit{vertex } s \textit{ matched in } M)}/|S|] = 2|M|/n.$$

Thus, we can output

$$\frac{n}{2|S|} \cdot \sum_{s \in S} 1_{(\textit{vertex } s \textit{ matched in } M)}$$

as an unbiased estimator of $|M|$, which by Lemma 1.58, we know is within a 2-factor of the size of the maximum matching.

The key thing that remains to be discussed is how do we deduct whether a given randomly sampled node s is matched in M or not, without running the entire maximal matching algorithm. Ideally, we should just check a few things around s and be able to infer whether s is matched in M or not. In particular, we will check each of the edges e incident on s separately, to see whether $e \in M$. Once we discuss this part, we will afterward explain how to set the number of sampled vertices S to obtain a desired degree of certainty and accuracy for this estimator, using basic concentration of measure arguments (e.g. Chernoff bound).

Checking $e \in M$ for One Edge, and its Query Complexity

Imagine the following procedure. Instead of running the entire random greedy maximal matching procedure to figure out whether $e \in M$ or not, we do something simpler, which should get the same answer: We determine the random value r_e and also all the values $r_{e'}$ for edges e' incident on e. We then *recursively* check whether any of these edges e' for which $r_{e'} < r_e$ is in the matching M or not. If none of them is in the matching, then e is in the matching. Clearly, if we use the same random numbers r_e as in the procedure explained above, we can deduce whether $e \in M$ or not, when we run the entire algorithm.

1.8.4 Analysis

The only question is, what is the query complexity of this recursive procedure. A priori, we might end up opening another recursion branch on each neighboring edge and thus we may conceivably end up with a query complexity up to $O(m)$. However, that is not likely. We next prove that for a given edge e, the expected query complexity of the recursive procedure is upper bounded to $2^{O(\Delta)}$, which is independent of n.

Lemma 1.59. *The expected query complexity of the algorithm for an arbitrary given edge e, which is selected independent of the random values r'_e for $e' \in E$, is at most $2^{O(\Delta)}$.*

Proof. Let $Q(x)$ be the maximum expected number of queries for any edge e, conditioned on $r_e = x$. Notice that $x \leqslant y$ implies $Q(x) \leqslant Q(y)$. Let $e_1, \dots, e_\ell$ be the edges incident on e. Notice that $\ell \leqslant 2\Delta - 2$. We claim that

$$Q(x) \leqslant (2\Delta - 2) + \sum_{i=1}^{\ell} \Pr[r_{e_i} < x] \cdot \mathbb{E}[Q(r_{e_i}) | r_{e_i} < x].$$

This is because, we first read all the up to $2\Delta - 2$ incident edges and then, for each neighboring edge e_i with $r_{e_i} < r_e$, we start a new recursion at e_i, which we know in expectation will take at most $Q(r_{e_i})$ queries. What remain is to solve this recursive relation $Q(x)$ and obtain an upper bound on it.

To simply the task of upper bounding this recursive inequality, we *discretize* it in some sense, that is, we upper bound it in only a bounded number of (well-spread) places. For any $j \in \{1, \dots, 4\Delta\}$, by setting $x = \frac{j}{4\Delta}$ in the above inequality, we have

$$\begin{aligned} Q(\frac{j}{4\Delta}) &< 2\Delta + \sum_{i=1}^{2\Delta} \Pr[r_{e_i} < \frac{j}{4\Delta}] \cdot \mathbb{E}[Q(r_{e_i}) | r_{e_i} < \frac{j}{4\Delta}] \\ &\leqslant 2\Delta + \sum_{i=1}^{2\Delta} \sum_{k=1}^{j} \mathbb{E}\big[Q(r_{e_i}) | r_{e_i} \in [\frac{k-1}{4\Delta}, \frac{k}{4\Delta})\big] \cdot \Pr[r_{e_i} \in [\frac{k-1}{4\Delta}, \frac{k}{4\Delta})] \\ &\leqslant 2\Delta + \sum_{i=1}^{2\Delta} \sum_{k=1}^{j} Q(\frac{k}{4\Delta}) \cdot \frac{1}{4\Delta} \leqslant 2\Delta + \frac{1}{2} \sum_{k=1}^{j} Q(\frac{k}{4\Delta}). \end{aligned}$$

By rearranging the terms around this inequality, we arrive at the following simpler recursion:

$$Q(\frac{j}{4\Delta}) \leqslant 4\Delta + \sum_{k=1}^{j-1} Q(\frac{k}{4\Delta}).$$

Given this recursion, by a simple induction on j, we can prove that $Q(\frac{j}{4\Delta}) \leqslant 4\Delta(2^j - 1)$ (why?). Hence, $Q(r_e) \leqslant Q(1) = Q(\frac{4\Delta}{4\Delta}) \leqslant 4\Delta(2^{4\Delta} - 1) = 2^{O(\Delta)}$. □

Observation 1.60. *The overall expected query complexity for checking a set S of nodes, to see whether they are matched in M or not, is at most $|S| \cdot \Delta 2^{O(\Delta)} = |S| \cdot 2^{O(\Delta)}$.*

This essentially determines the query complexity of our algorithm. Even for a small sample set S, this is an unbiased estimator. However, we usually would like to say that the estimator has a good probability to be a good approximation of its target value. Next, we discuss how by picking a large enough sample set size $k = |S|$, we can satisfy these desires.

Adjusting the Sample Set For the Desired Accuracy and Certainty

Lemma 1.61. *For any certainty parameter $\delta \in [0, 0.25]$ and any precision parameter $\epsilon > 0$, suppose that we choose a set S of $k = \frac{20\Delta \log 1/\delta}{\epsilon^2}$ nodes at random, with replacement, and check whether each $s \in S$ is matched in the maximal matching M or not. Then, the function*

$$\frac{n}{2|S|} \cdot \sum_{s \in S} 1_{(vertex\ s\ matched\ in\ M)}$$

provides a $(1+\epsilon)$ approximation of the size of maximal matching, with probability at least $1-\delta$. Hence, this is a $2(1+\epsilon)$ approximation of the size of maximum matching.

Notice that by the above observation, the expected query complexity of our algorithm becomes $k2^{O(\Delta)} = 2^{O(\Delta)} \cdot \frac{\log 1/\delta}{\epsilon^2}$.

Proof of Lemma 1.61. Define X_i to be the random variable that is equal to 1 if the i^{th} node in our sample set S is matched in M, and is equal to 0 otherwise. Notice that $\Pr[X_i = 1] = \frac{2|M|}{n}$. Hence, $\mu = \mathbb{E}[\sum_{i=1}^{k} X_k] = \sum_{i=1}^{k} \mathbb{E}[X_i] = \sum_{i=1}^{k} \Pr[X_i = 1] \cdot 1 = k\frac{2|M|}{n}$. Therefore, by Chernoff bound, the probability that $X = \sum_{i=1}^{k} X_k$ deviates by more than a $(1+\epsilon)$ factor from its expectation μ is at most

$$2e^{-\epsilon^2 \mu/3} = 2e^{-\epsilon^2 k \cdot 2|M|/(3n)} = 2e^{-\epsilon^2 \frac{20\Delta \log 1/\delta}{\epsilon^2} 2|M|/(3n)} = 2e^{-10\frac{\Delta|M|}{3n} \cdot \log 1/\delta} \leqslant \delta.$$

The last inequality uses $|M| \geqslant \frac{n}{4\Delta}$. This fact holds because we have assumed that the graph has no isolated edge, and thus it has at least $n/2$ edges, and moreover, each edge in the maximal matching M can hit at most 2Δ edges (including itself) and all edges must be hit. □

Remark The algorithm presented here is (a streamlined variant of) a result of Nguyen and Onak [NO08]. See their paper for how this technique can be used for a range of other approximation problems. In the exercises of this lecture, we will see an alternative method for computing an approximation of maximum matching, which has a better query complexity as a function of Δ, but with a slightly worse dependency on the precision parameter ϵ.

1.9 Exercises

Exercise 1.1. *In Lemma 1.5, we saw a single-round algorithm for reducing the number of colors exponentially. Here, we discuss another such method, which transforms any* k*-coloring of any rooted-tree to a* $2\log k$*-coloring, so long as* $k \geqslant C_0$ *for a constant* C_0*.*

The method works as follows. Let each node u *send its color* $\phi_{old}(u)$ *to its children. Now, each node* v *computes its new color* $\phi_{new}(v)$ *as follows: Consider the binary representation of* $\phi_{old}(v)$ *and* $\phi_{old}(u)$*, where* u *is the parent of* v*. Notice that each of these is a* $\log_2 k$*-bit value. Let* i_v *be the smallest index* i *such that the binary representations of* $\phi_{old}(v)$ *and* $\phi_{old}(u)$ *differ in the* i^{th} *bit. Let* b_v *be the* i_v^{th} *bit of* $\phi_{old}(v)$*. Define* $\phi_{new}(v) = (i_v, b_v)$*. Prove that* $\phi_{old}(v)$ *is well-defined, and that it is a proper* $(2\log k)$*-coloring.*

Exercise 1.2. *Here, we use the concept of cover free families, as defined in Definition 1.18, to obtain an encoding that allows us to recover information after superimposition. That is, we will be able to decode even if* k *of the codewords are* superimposed *and we only have the resulting bit-wise OR.*

More concretely, we want a function $\mathrm{Enc} : \{0,1\}^{\log n} \to \{0,1\}^m$ *— that encodes* n *possibilities using* m*-bit strings for* $m \geqslant \log_2 n$ *— such that the following property is satisfied:* $\forall S, S' \subseteq \{1,\dots,n\}$ *such that* $|S| \leqslant k$ *and* $|S'| \leqslant k$*, we have that* $\bigvee_{i\in S}\mathrm{Enc}(i) \neq \bigvee_{i\in S'}\mathrm{Enc}(i)$*. Here* $\bigvee$ *denotes the bit-wise OR operation.*

Present such an encoding function, with a small m*, that depends on* n *and* k*. What is the best* m *that you can achieve?*

Exercise 1.3. *This exercise has two parts:*

(A) Design a single-round algorithm that transforms any given k*-coloring of a graph with maximum degree* Δ *into a* k'*-coloring for* $k' = k - \lceil \frac{k}{2(\Delta+1)} \rceil$*, assuming* $k' \geqslant \Delta + 1$*.*

(B) Use repetitions of this single-round algorithm, in combination with the $O(\log^* n)$*-round* $O(\Delta^2)$*-coloring of Theorem 1.16, to obtain an* $O(\Delta\log\Delta + \log^* n)$*-round* $(\Delta+1)$*-coloring algorithm.*

Exercise 1.4. *Here, we see yet another deterministic method for computing a* $(\Delta+1)$*-coloring in* $O(\Delta\log\Delta + \log^* n)$ *rounds. First, using Theorem 1.16, we compute an* $O(\Delta^2)$*-coloring* ϕ_{old} *in* $O(\log^* n)$ *rounds. What remains is to transform this into a* $(\Delta+1)$*-coloring, in* $O(\Delta\log\Delta)$ *additional rounds.*

The current $O(\Delta^2)$*-coloring* ϕ_{old} *can be written using* $C\log\Delta$ *bits, assuming a sufficiently large constant* C*. This bit complexity will be the parameter of our* *recursion. Partition G into two vertex-disjoint subgraphs G_0 and G_1, based on the most significant bit in the color ϕ_{old}. Notice that each of G_0 and G_1 inherits a coloring with $C \log \Delta - 1$ bits. Solve the $\Delta + 1$ coloring problem in each of these independently and recursively. Then, we need to merge these colors, into a $\Delta + 1$ coloring for the whole graph.*

- *(A) Explain an $O(\Delta)$-round algorithm, as well its correctness proof, that once the independent $(\Delta + 1)$-colorings of G_0 and G_1 are finished, updates only the colors of G_1 vertices to ensure that the overall coloring is a proper $(\Delta + 1)$-coloring of $G = G_0 \cup G_1$.*
- *(B) Provide a recursive time-complexity analysis that proves that overall, the recursive method takes $O(\Delta \log \Delta)$ rounds.*

Exercise 1.5. *In this exercise, we prove a lower bound of $\Omega(\log n / \log \log n)$ on the round complexity of computing a* locally-minimal coloring. *For a graph $G = (V, E)$, a coloring $\phi : V \to \{1, 2, ..., Q\}$ is called* locally-minimal *if it is a proper coloring, meaning that no two adjacent vertices v and u have $\phi(v) = \phi(u)$, and moreover, for each node v colored with color $q = \phi(v) \in \{1, 2, ..., Q\}$, all colors 1 to $q - 1$ are used in the neighborhood of v. That is, for each $i \in \{1, \dots, q-1\}$, there exists a neighbor u of v such that $\phi(u) = i$.*

To prove the lower bound, we will use a classic graph-theoretic result of Erdős [Erd59]. Recall that the girth of a graph is the length of its shortest cycle, and the chromatic number of a graph is the smallest number of colors required in any proper coloring of the graph.

Theorem 1.62 (Erdős [Erd59]). *For any sufficiently large n, there exists an n-node graph G^*_n with girth $g(G^*_n) \geqslant \frac{\log n}{4 \log \log n}$ and chromatic number $\chi(G^*_n) \geqslant \frac{\log n}{4 \log \log n}$.*

- *(A) Prove that in any locally-minimal coloring $\phi : V \to \{1, 2, ..., Q\}$ of a tree $T = (V, E)$ with diameter d — i.e., where the distance between any two nodes is at most d — no node v can receive a color $\phi(v) > d + 1$.*
- *(B) Suppose towards contradiction that there exists a deterministic algorithm $\mathcal{A}$ that computes a locally-minimal coloring of any n-node graph in at most $\frac{\log n}{8 \log \log n} - 1$ rounds. Prove that when we run $\mathcal{A}$ on the graph G^*_n, it produces a (locally-minimal) coloring with at most $Q = \frac{\log n}{4 \log \log n} - 1$ colors. For this, you should use part (1b) and the fact that G^*_n has girth $g(G^*_n) \geqslant \frac{\log n}{4 \log \log n}$.*
- *(C) Conclude that any locally-minimal coloring algorithm needs at least $\frac{\log n}{8 \log \log n}$ rounds on some n-node graph.*

Exercise 1.6. *In this exercise, we see a simple and efficient sequential algorithm that, for every* n*-node graph* G*, computes an* $(\mathcal{C}, \mathcal{D})$ *(strong-diameter) network decomposition for* $\mathcal{C} = O(\log n)$ *and* $\mathcal{D} = O(\log n)$*.*

We determine the blocks G_1*,* G_2*, ...,* $G_{\mathcal{C}}$ *of network decomposition one by one, in* C *phases. Consider phase* i *and the graph* $G \setminus \big(\cup_{j=1}^{i-1} G_j\big)$ *remaining after the first* $i-1$ *phases which defined the first* i *blocks* $G_1, \dots, G_{i-1}$*. To define the next block, we repeatedly perform a ball carving starting from arbitrary nodes, until all nodes of* $G \setminus \big(\cup_{j=1}^{i-1} G_j\big)$ *are removed. This ball carving process works as follows: consider an arbitrary node* $v \in G \setminus \big(\cup_{j=1}^{i} G_j\big)$ *and consider gradually growing a ball around* v*, hop by hop. In the* k^{th} *step, the ball* $B_k(v)$ *is simply the set all nodes within distance* k *of* v *in the remaining graph. In the very first step that the ball does not grow by more than a* 2 *factor — i.e., smallest value of* k *for which* $|B_{k+1}(v)|/|B_k(v)| \leqslant 2$ *— we stop the ball growing. Then, we carve out the inside of this ball — i.e., all nodes in* $B_k(v)$ *— and define them to be a cluster of* G_i*. Hence, these nodes are added to* G_i*. Moreover, we remove all boundary nodes of this ball —i.e., those of* $B_{k+1}(v) \setminus B_k(v)$*—and from the graph considered for the rest of this phase. These nodes will never be put in* G_i*. We will bring them back in the next phases, so that they get clustered in the future phases. Then, we repeat a similar ball carving starting at an arbitrary other node* v' *in the remaining graph. We continue a similar ball carving until all nodes are removed. This finishes the description of phase* i*. Once no node remains in this graph, we move to the next phase. The algorithm terminates once all nodes have been clustered.*

Prove the following properties:

(A) Each cluster defined in the above process has diameter at most $O(\log n)$*. In particular, for each ball that we carve, the related radius* k *is at most* $O(\log n)$*.*

(B) In each phase i*, the number of nodes that we cluster —and thus put in* G_i *— is at least* $1/2$ *of the nodes of* $G \setminus \big(\cup_{j=1}^{i} G_j\big)$*.*

(C) Conclude that the process terminates in at most $O(\log n)$ *phases, which means that the network decomposition has at most* $O(\log n)$ *blocks.*

Exercise 1.7. *Explain how given a* (C, D) *network decomposition of graph* G*, a maximal independent set can be computed in* $O(CD)$ *rounds.*

Exercise 1.8. *Suppose that you are given a deterministic algorithm* $\mathcal{ALG}$ *that on any* N*-node graph, computes a* $(C(N), D(N))$ *weak-diameter network decomposition, in* $T(N)$ *rounds. Develop a new deterministic distributed algorithm that computes an* $(\mathcal{C}, \mathcal{D})$ *strong-diameter network decomposition in any* n*-node network, in* $T(n) \cdot O(\log n) + O(C(n) \cdot D(n) \cdot \log^2 n)$ *rounds.*

Exercise 1.9. *Prove Lemma 1.32. That is, show that in the randomized network decomposition provided in Section 1.5.2, with high probability, the maximum cluster diameter is at most* $O(\log n/\varepsilon)$.

Exercise 1.10. *Improve the round complexity of the algorithm stated in Theorem 1.31 to* $O(\log n)$ *rounds.*

Exercise 1.11. *In this exercise, we prove that the network decomposition obtained in Theorem 1.31 has the nearly best possible parameters. As we discussed above in Theorem 1.62, it is known that there are* n*-node graphs that have girth*[14] $\Omega(\log n/\log\log n)$ *and chromatic number* $\Omega(\log n)$*[AS04, Erd59]. Use this fact to argue that on these graphs, an* $(o(\log n), o(\log n/\log\log n))$ *network decomposition does not exist.*

Exercise 1.12. *Given an* n*-node undirected graph* $G = (V, E)$*, a* $d(n)$-diameter ordering *of* G *is a one-to-one labeling* $f : V \to \{1, 2, \dots, n\}$ *of vertices such that for any path* $P = v_1, v_2, \dots, v_p$ *on which the labels* $f(v_i)$ *are monotonically increasing, any two nodes* $v_i, v_j \in P$ *have* $\mathrm{dist}_G(v_i, v_j) \leqslant d(n)$.

Use the network decomposition of Theorem 1.31 to argue that each n*-node graph has an* $O(\log^2 n)$*-diameter ordering.*

Exercise 1.13. *Consider the following simple* 1*-round randomized algorithm: each node* v *picks a random real number* $r_v \in [0, 1]$ *and then,* v *joins a set* S *if its random number is a local minima, that is, if* $r_v < r_u$ *for all neighbors* u *of* v*. Prove that, with high probability, the set* S *is a* $(2, O(\log n))$*-ruling set.*

Exercise 1.14. *Consider a regularized variant of Luby's MIS algorithm, as follows: The algorithm consists of* $\log \Delta + 1$ *phases, each made of* $O(\log n)$ *consecutive rounds. Here* Δ *denotes the maximum degree in the graph. In each round of the* i^{th} *phase, each remaining node is marked with probability* $\frac{2^i}{10\Delta}$*. Different nodes are marked independently. Then marked nodes who do not have any marked neighbor are added to the MIS set, and removed from the graph along with their neighbors. If at any time, a node* v *becomes isolated and none of its neighbors remain, then* v *is also added to the MIS and is removed from the graph.*

(A) *Argue that the set of vertices added to the MIS is always an* independent set.

[14] Recall that the girth of a graph is the length of its shortest cycle.

(B) *Prove that with high probability, by the end of the* i^{th} *phase, in the remaining graph each node has degree at most* $\frac{\Delta}{2^i}$.

(C) *Conclude that the set of vertices added to the MIS is a* maximal *independent set, with high probability.*

Exercise 1.15. *Consider the following simple randomized* $\Delta + 1$ *coloring algorithm: Per round, each node selects one of the colors not already* taken away *by its neighbors, at random. Then, if* v *selected a color and none of its neighbors selected the same color in that round,* v *gets colored with this color and takes this color away permanently. That is, none of the neighbors of* v *will select this color in any of the future rounds.*

(A) *Prove that in the first round, each node has at least a constant probability of being colored.*

(B) *Prove that per round, each remaining node has at least a constant probability of being colored. This is somewhat similar to part (B) but requires much more care, because after the first round, the colors remaining for a node or for its neighbors can be quite different.*

(C) *Conclude that within* $O(\log n)$ *rounds, all nodes are colored, with high probability.*

Chapter 2

Global Problems

2.1 Introduction & the CONGEST Model

In this chapter, we discuss distributed algorithms for some of the fundamental *global graph problems*, such as minimum spanning tree and minimum cut computation. These are global problems in the sense that the solution can depend on pieces of information that are far away, e.g., even up to $\Theta(D)$-hops away, where D denotes the network diameter. Other central examples, which we will not get to discuss here, include computing single source shortest paths (SSSP), or all pairs shortest paths (APSP), minimum-cost subgraphs with various properties (e.g., connected dominating set or k-edge-connected spanning subgraph), etc.

In studying global problems, we work with the $\mathsf{CONGEST}$ model, which is a variant of the LOCAL model where we take the bandwidth limitations into account.

Definition 2.1. *(The CONGEST model) We consider an arbitrary n-node graph $G = (V, E)$ where $V = \{1, 2, \dots, n\}$, which abstracts the communication network. Unless noted otherwise, G is a simple, undirected, and unweighted graph. There is one process on each node $v \in V$ of the network. At the beginning, the processes do not know the graph G, except for knowing*[1] *n, and their own unique identifier in $\{1, 2, \dots, n\}$. The algorithms work in synchronous rounds. Per round, each node/process performs some computation based on its own knowledge, and then sends one $B - \mathrm{bit}$ message to each of its neighbors. Usually, we assume that $B = O(\log n)$, which, e.g., implies that each message can describe constant many edges or vertices of the network. A node can send different messages to different neighbors. At the end of the round, each node receives the messages send to it by its neighbors in that round. In each graph problem in this model, we require that each node learns its own part of the output, e.g., whether each of its edges is in a computed tree or not.*

A first-order summary of global network optimization problems Distributed algorithms for global network optimization problems have a long and rich history. A first-order summary of the state of the art is that, for many of the fundamental problems—including minimum spanning tree, minimum cut, maximum flow Approximation, and shortest path computations—there is a lower bound of $\tilde{\Omega}(D + \sqrt{n})$ rounds [PR99, Elk04, DSHK+11], in general graphs. In particular, there is a graph of fairly small diameter $D = O(\log n)$ in which solving these problems, or often even obtaining a non-trivial approximation, requires $\tilde{\Omega}(\sqrt{n})$ rounds. Over the past decade, there has been also a flurry of positive results that provide algo-

[1]Most often, the algorithms will use only the assumption that nodes know an upper bound N on n such that $N \in [n, n^c]$ for a small constant $c \geqslant 1$.

rithms with a round complexity of $\tilde{O}(D + \sqrt{n})$, or other bounds approaching it, for a number of these problems. See for instance [LPS13, GK13, Nan14, LPS14, CHGK14, NS14, GL14, Gha14, GKK+15, GU15, BFKL16, EN16, HKN19, Elk17, GL18a, FN18, Dor18, DG19, DHIN19, DEMN20]. In the next two sections, as examples of these upper bounds, and to showcase some of the basic techniques used throughout, we discuss algorithms for the minimum spanning tree problem and for the minimum cut approximation problem.

2.2 Minimum Spanning Tree

In this section, we discuss a near-optimal distributed algorithm in the CONGEST model for the problem of computing a minimum spanning tree (MST). We note that, over the years, the problem of computing an MST has turned out to have a central role in distributed algorithms for network optimization problems, significantly more central than its role in the centralized algorithms domain. The upper and lower bound techniques for the MST problem are used frequently in solving other distributed network optimization problems. What we see in this section is based on the work of Kutten and Peleg [KP95], although we deviate from their approach in a few parts.

2.2.1 MST: The Algorithm Outline

The algorithm we describe follows the outline of Boruvka's MST [NMN01] from 1926, though with some small changes. In particular, we have $O(\log n)$ phases. During these phases, we gradually grow a forest until we reach a spanning tree. We start with the trivial forest where each node forms its own component of the forest, that is, each node is one separate component in our partition of G.

In each phase, each component S_i will have a leader node $s_i \in S_i$, and moreover, the leader will know the size of its component. Each component S_i suggests a merge along the edge with exactly one endpoint in S_i that has the smallest weight among such edges. This is called the *minimum weight outgoing edge (MWOE)*. Recall from the second lecture (lemma 2.17) that all such edges belong to MST[2]. We soon explain how to compute these min-weight outgoing edges, one per part, using shortcuts. Let us for now

[2]This assumes that the edge-weights are unique, which is effectively without loss of generality, because we can append the identifier of the edge—composed of the identifiers of its two endpoints—to its weight in a manner that makes the edge weights unique, and guarantees that the MST according to the new weights is one of the MSTs according to the original weights.

continue with the high-level explanation of how to use these edges to merge parts.

Let N be the current number of connected components of the forest. If $N = 1$, we are done already. Otherwise, each component suggests one merge edge. Each edge might be suggested by two of its endpoints, so we have at least $N/2$ suggested edges in total. We add these edges to the forest and thus, effectively, merge the connected components at their two endpoints. Hence, the number of components shrinks to at most $N/2$. After $\log n$ iterations, the number of connected components is down to 1, which means we have reached a spanning tree.

What remains is to explain how to find the merge edges, and how to perform the merges. We first discuss the process of computing the minimum-weight outgoing edges of one phase in $O(D + \sqrt{n})$ rounds. Then, we will discuss how to perform the merges in the same round complexity.

2.2.2 Computing Min-Weight Outgoing Edges

Our objective is to let each node know the minimum weight outgoing edge of its component. More concretely, let each node v set $c(v)$ to be the minimum-weight outgoing edge among edges incident to v. The objective is that each node $v \in S_i$ learns the weight of the minimum-weight outgoing edge among edges all edges incident on component S_i. Notice that node v can easily find $c(v)$ by first receiving from all neighbors the component leader IDs of their components and then only considering the smallest of those edges having the other endpoint in a different component. We handle the components in two categories of small and large, depending on whether the component has at least $\sqrt{n}$ vertices or not.

Small Components Consider a single small component S_i, which means this component has no more than $\sqrt{n}$ vertices. Then, in this component, each node v starts with its own minimum-weight outgoing edge and its weight $c(v)$. Then, we perform a convergecast (or simply minimum flooding) on the BFS tree of this component S_i. This convergecast goes from the leaves to the root, maintaining the minimum value seen, and thus eventually delivering the minimum-weight outgoing edge to the component leader. The information about this edge can be delivered to all nodes of the component by a broadcast from the root to the leaves.

Large Components There are at most $n/\sqrt{n} = \sqrt{n}$ large parts, as each of them has at least $\sqrt{n}$ vertices. Since the number of large components is relatively small, we can handle all these components by performing their communications on the BFS of the whole graph G, simultaneously. Using standard pipelining techniques[Peloo], we can compute the minimum-weight outgoing edges of all these $\sqrt{n}$ components in $O(D+\sqrt{n})$ rounds.

A more general treatment

The above division to small and large categories, and then the rules for where each of these should communicate, leads to an $O(D+\sqrt{n})$-round algorithm for one phase and thus an $\tilde{O}(D+\sqrt{n})$-round algorithm for MST. This is nearly-optimal in the worst case. However, in many graphs families of interest, this would be quite far from desirable bounds. In the following, and mostly as side remarks, we introduce a graph-theoretic notion of *low-congestion shortcuts* and briefly outline how this notion leads to more efficient algorithms, as well as a simple and clean unification of many methods.

Definition 2.2 (Low-Congestion Shortcuts). *Consider a graph $G = (V, E)$ and a partition of V into disjoint subsets $S_1, \dots, S_N \subset V$, each inducing a connected subgraph $G[S_i]$. We define an α-congestion shortcut with dilation β to be a set of subgraphs $H_1, \dots, H_N \subset G$, one for each set S_i, such that:*

(1) For each i, the diameter of the subgraph $G[S_i] + H_i$ is at most β.

(2) For each $e \in E$, the number of subgraphs $G[S_i] + H_i$ containing e is at most α.

Theorem 2.3. *Suppose that the graph family $\mathcal{G}$ is such that for each graph $G \in \mathcal{G}$, and any partition of G into vertex-disjoint connected subsets $S_1, \dots, S_N$, we can find an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leqslant K$. Here, K can be a function of the family $\mathcal{G}$, and it can depend on n and D. Moreover, we assume such a low-congestion shortcut can be found in $\tilde{O}(T)$ rounds.*

Then, there is a randomized distributed MST algorithm that computes an MST in $\tilde{O}(T) + O(\alpha \log n + \beta \log^2 n) = \tilde{O}(T+K)$ rounds, with high probability, in any graph from the family $\mathcal{G}$.

It is not hard to see that the above rule for small and large components can be used to infer that any graph has a α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leqslant D + \sqrt{n}$, and thus leads to an $\tilde{O}(D+\sqrt{n})$-round MST algorithm for general graphs. There are a number of graph families where the question, and especially its graph-theoretic aspects, becomes much more interesting:

- For planar graphs and a few generalizations (e.g. bounded-genus graphs, bounded pathwidth or treewidth grapths, and more generally any graph that does not include a dense minor), it has been shown[GH16, HIZ16a, HIZ16b, HHW18, HLZ18, GH20] that there always exists an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leqslant \tilde{O}(D)$ and moreover, such a shortcut can be found in $\tilde{O}(D)$ rounds. This leads to an $\tilde{O}(D)$-round algorithm for MST in planar and near-planar graphs. See [GH20] for the most general result and also the simplest proof.

- In Erdős-Renyi random graph $G_{n,p}$, where each of the possible $\binom{n}{2}$ edges is included with probability $p \geqslant \Omega(\log n/n)$ (that is, above the connectivity threshold), it has been shown that there always exists a low-congestion shortcut with $\max\{\alpha, \beta\} \leqslant \operatorname{poly}\log n$, and such a shortcut can be found in $2^{O(\sqrt{\log n})}$ rounds. That leads to an $2^{O(\sqrt{\log n})}$-round algorithm for MST in Erdős-Renyi random graphs. See [GKS17, GL18b] for this result and extensions to much broader graph families (those with small random walk mixing time).

2.2.3 Back to Worst-Case Graphs, Merging Components

A small change in Boruvka's outline to have low-depth merges We restrict the merges to be *star* shapes, using a simple random coin idea: toss a random coin per component and then allow only merges centered on head-parts, each accepting incoming suggested merge-edges from tail-parts. The leader of this head-component becomes the leader of the merged new part. In exercise 2 of today's lecture, we see that, albeit this slightly slowed down probabilistic merging process, still after $O(\log n)$ phases, with high probability, we reach a tree.

What we need to compute for a merge We need to make all nodes learn, besides the minimum-weight outgoing edge of their component, three extra pieces of information: (1) the coin tossed by their component leader, (2) the ID of their new component leader, (3) the size of the new component. We next explain how to perform each of these steps in $O(D + \sqrt{n})$ rounds.

- Item (1)—which is to let each node know the coin toss of its component leader—is by a simple small change to the messages sent in computing the min-weight outgoing edge: now the message starting at the root also carries the random bit flipped by the leader.

- Item (2)—which is to let each node know its new component leader ID—is performed as follows: We define the component leader ID to be the leader of the center component of the merge, who had a head coin. This ID is already delivered to the physical endpoint of the merge edge in the tail part.

Hence, within the tail component, all that we need to do is that one node knows the ID of the new leader and we want all nodes to have it. If the component was small, we can do it directly in $O(\sqrt{n})$ rounds, inside the component. For large components, which there are only at most $\sqrt{n}$ of them, we can broadcast the their new ID leader, which is known to the physical endpoint of their merge edge, to all nodes of the graph in $O(D + \sqrt{n})$ rounds.

- Item (3)—which is to let each node know the size of its component—can be performed similar to (2). First, in the center-of-merge head component, the physical endpoints of the merge can receive from their other endpoints the sizes of the merging tail components. Then, within this head component, we can compute the new component size by performing a simple converge-case, if the component is small, and by doing it through the global BFS tree, for large components. At the end, this information can be passed to all vertices of the new component, by first delivering it to all nodes of the head component, then passing it through the physical edges to the tail components, and then spreading it in the tail components.

2.3 Minimum Cut

In this lecture, we explain a distributed algorithm that computes a $(2 + \epsilon)$ approximation of the minimum cut, for any constant $\epsilon > 0$, in $O(k(D + \sqrt{n}) \log^3 n)$ rounds. Here, k denotes the size of the minimum cut, i.e., the minimum number of edges whose removal disconnects the graph. The algorithm should present a nonempty subset of vertices S such that the number of edges connecting nodes of S to nodes of $V \setminus S$ is at most $(2 + \epsilon)k$. As usual, we work in the standard synchronous message-passing model of distributed computing where the network is abstracted as an n-node connected graph with diameter D and per round each node can send $O(\log n)$ bits to each of its neighbors.

A number of remarks are in order:

- First, the k factor in the round complexity can be removed and replaced with $O(\log n)$, using a sampling idea which reduces the minimum cut to $O(\log n)$, while preserving the sizes of all cuts approximately.

- Second, the algorithm can be generalized to weighted graphs where each edge has a weight in $\{1, 2, \ldots, \text{poly}(n)\}$ and the size of a cut is the summation of the weights of the edges whose removal disconnects the graph. For this generalization, we simply view each edge of w as w many parallel edges.

- Third, the resulting $\tilde{O}(D+\sqrt{n})$ round complexity is nearly optimal, as a known lower bound shows that any distributed algorithm for (any non-trivial) approximation of the minimum cut requires at least $\tilde{\Omega}(D+\sqrt{n})$ rounds. See Ghaffari and Kuhn [GK13] for details of the above three points.

- Fourth, a $(1+\epsilon)$ approximation algorithms with a round complexity of $\tilde{O}(D+\sqrt{n})$ is known to a work of Nanongkai and Su [NS14]; we will not cover it in this lecture.

- Finally, in a recent breakthrough, Daga et al. [DHNS19] presented the first distributed algorithm that computes the exact minimum cut in a sublinear number of rounds, particularly in $\tilde{O}(n^{1-1/353}D^{1/353}+n^{1-1/706})$ rounds in unweighted graphs. More recently, Ghaffari and Nowicki [GTN20] presented a different algorithm that improved the round complexity further to $\tilde{O}(n^{1-1/9}D^{1/9}+n^{1-1/18})$.

2.3.1 Sparse Certificates for Connectivity

Before presenting the algorithm for minimum cut approximation, first we introduce the concept of *sparse certificates*. In a very rough sense, this is a subgraph that maintains the size of the minimum cut while reducing the number of edges. Notice that any graph that has minimum-cut size at least k must have at least $nk/2$ edges. This is because each node in such a graph must have degree at least k, as otherwise removed the edges adjacent to that node would yield a cut with size less than k. A sparse certificate aims to reduce the number of the edges in any graph to something close to this threshold of $nk/2$, while maintaining the minimum cut size.

Definition 2.4. *For a graph* $G=(V,E)$, *we call a spanning subgraph* $H=(V,E_H)$ *a* sparse certificate of G for connectivity k' *if the following two conditions are satisfied:*

(1) For each nonempty subset $S \subset V$ *of vertices, we have* $\mathrm{cut}_H(S, V\setminus S) \geqslant \min\{k', \mathrm{cut}_G(S, V\setminus S)\}$. *Here,* $\mathrm{cut}_H(S, V\setminus S)$ *denotes the number of edges in* H *that have one endpoint in* S *and the other endpoint in* $V\setminus S$. *Similarly,* $\mathrm{cut}_G(S, V\setminus S)$ *denotes the number of edges in* G *that have one endpoint in* S *and the other endpoint in* $V\setminus S$.

(2) Subgraph H *has at most* $|nk'|$ *edges.*

Example: In a connected graph G, any spanning tree T is a sparse certificate for connectivity $k' = 1$. It clearly satisfies condition (2), because it has $n-1$ edges, and it satisfies condition (1) as well because $\mathrm{Cut}_T(S, V \setminus S) \geqslant 1$. More generally, if we do not assume G to be connected, any maximal spanning forest is a sparse certificate for connectivity $k' = 1$.

Computing a Sparse Certificate for Connectivity k' (Algorithm Outline): As the above example reveals, there is a simple and natural algorithm for computing a sparse certificate for connectivity k'. Here, we discuss the algorithm outline from a centralized viewpoint. We later discuss a distributed algorithm that implement this outline, efficiently. The algorithm works in k' iterations, as follows: In iteration i, compute a maximal spanning forest F_i of G and then update $G \leftarrow G \setminus F_i$. Here, $G \setminus F_i$ means we remove from G all edges of F_i. At the end of algorithm, the union $H = \bigcup_{i=1}^{k'} F_i$ of the forests computed in the k' iterations is our desired sparse certificate for connectivity k'.

Lemma 2.5. *In the above algorithm,* $H = \bigcup_{i=1}^{k'} F_i$ *is a sparse certificate for connectivity* k'.

Proof. Since the number of edges in each forest F_i is at most $n-1$, graph H has at most $k(n-1) \leqslant kn$ edges and thus it clearly satisfies condition (2). To see that it also satisfies condition (1), let us focus on an arbitrary cut $(S, V \setminus S)$. In each iteration i, so long as at least one edge of G remains in the cut $(S, V \setminus S)$, the forest F_i will include at least one edge from $(S, V \setminus S)$. Notice that it might also include more than edge. Hence, the number of edges of H in cut $(S, V \setminus S)$ is either at least k, or the algorithm exhausted all the edges of this cut and therefore the number is equal to $\mathrm{cut}_G(S, V \setminus S)$. □

Distributed Implementation of the Above Outline: A naive idea for implementing the above outline would be to compute each F_i separately, in the remaining graph. However, this might be quite inefficient as the remaining graph might have a very large diameter. To go around this issue, we use some artificial edge weights to implement the outline. In particular, initially, we set the weight of each edge to be 1. Then, in each iteration i, we compute a minimum-weight spanning tree T_i and define F_i to be all edges of T_i that have weight 1. Then, instead of removing these edges from the graph, we simply set their weight to be ∞ (or realistically, just n^2). Computing each minimum-weight spanning tree T_i can be done in $O((D + \sqrt{n}) \log n)$ rounds, using the MST algorithm that we discussed in the previous lecture.

Lemma 2.6. *Consider the weighted version of* G *where we set the edge weight of* $G \setminus (\cup_{j=1}^{i-1} F_i)$ *to be* 1 *and the edge weights of* $(\cup_{j=1}^{i-1} F_i)$ *to be* n^2*. Then, the edges of weight* 1 *in the minimum-weight spanning tree* T_i *form a maximal spanning forest of the graph* $G \setminus (\cup_{j=1}^{i-1} F_i)$*.*

Proof Idea. Informally, T_i will try to include as many as possible of weight 1 edges before including any weight n^2 edges. Formalizing this into a proof is left as an exercise. □

2.3.2 Approximation Algorithm for Minimum Cut

We now describe the algorithm for computing a $(2+\varepsilon)$ approximation of the minimum cut. For now, we assume that the value k of the minimum cut size is known. The algorithm will compute a subset $S \subset V$ such that $cut_G(S, V \setminus S) \leqslant (2+\varepsilon)k$.

Intuitive Description of the Algorithm: First, we compute a sparse certificate H for connectivity $k' = (1+\varepsilon/10)k$. Notice that this can be done in $O(k(D+\sqrt{n})\log n)$ rounds, as we discussed above. Now, think about the auxiliary graph H' where we *contract* each edge of $G \setminus H$, that is, we unify the two endpoint nodes into one node. Notice two properties: (A) Edges of H' are exactly edges of H (while the nodes where contracted) and therefore H' has at most $nk' = (1+\varepsilon/10)nk$ edges. Moreover, (B) every cut of H' has size at least k, and every cut of H' that has size less than k' must have had size less than k' also in G. Hence, H' has minimum cut size k and any minimum cut of it corresponds to a minimum cut of G (and vice versa). Now, H' might be in one of the following two cases. In each case, we can handle the problem differently.

(1) Suppose that H' has at least $n/(1+\varepsilon/5)$ nodes. In this case, the average degree in H' is at most $\frac{2(1+\varepsilon/10)nk}{n/(1+\varepsilon/5)}$ which implies that H' has at least one node with degree at most $2k(1+\varepsilon/10)(1+\varepsilon/5) \leqslant (2+\varepsilon)k$. This node in H' is the result of contracting some edges of G, and is therefore some subset S of vertices of G (which were contracted together). We get that the number of edges from S to $V \setminus S$ is at most $(2+\varepsilon)k$, that is, S is our desired approximate minimum cut. Hence, if we are in the case that H' has at least $n/(1+\varepsilon/5)$ nodes, then the problem is easy.

(2) Suppose that H' has less than $n/(1+\varepsilon/5)$ nodes. In this case, it is not so clear to identify a small cut. However, we have managed to make some progress because the number of nodes has reduced to $n/(1+\varepsilon/5)$, while we have preserved the minimum cut. Then, we can

recurse on this new graph H' by setting $G \leftarrow H'$, which has at most $n/(1+\varepsilon/5)$ vertices. Notice that after $O(\log_{1+\varepsilon/5} n)$ such recursions, the number of remaining nodes would drop to at most 2 and thus, there, we have found our approximate minimum cut.

Distributed Implementation The above intuitive description provides the outline of one level of recursion, in our algorithm. We now discuss how to implement it in a distributed setting. Suppose we are in a setting where we are working on a subgraph G' of G, identified by its edges. Initially, in the very first iteration, G' will be same as G. In each iteration, at the end of the iteration, if we are in case (2) and need to recurse, if we want to contract edges of $G \setminus H$, we would set $G' \leftarrow H$. Effectively, each connected component of $G \setminus G'$ corresponds to one node in our contracted structure.

The algorithm in this iteration of recursion works as follows: we set the weights of edges of $G \setminus G'$ to be 0 and weights of edges of G' to be 1. Then, we compute a sparse certificate H of G' for connectivity $k' = (1+\varepsilon/10)k$, as we described in the previous subsection (iteratively computing MSTs, taking one-weight edges as F_i, and increasing their weight to n^2). At the end of the sparse certificate computation, by the definition of a sparse certificate, subgraph H has a number of edges that is at most a k' factor of the number of connected components of $G \setminus G'$. Hence, we are virtually moving to a graph where all edges of $G \setminus H$ are contracted (weight 0) and no edge of H is contracted.

Then, each component of $G \setminus H$ computes the number of its edges in H, using the algorithm that we saw in the previous lecture where each component could compute the minimum of its edges — now, instead of minimum, we compute summation, but the algorithm is the same. This can be done in $O((D+\sqrt{n})\log n)$ rounds, as we saw last week. If for at least one of the components, the number of edges is at most $(2+\varepsilon)k$, then we have found our approximate minimum cut. Otherwise, we set $G' = H$ and recurse.

Final Remark — Removing the Assumption of Known k: In the above, we assumed that we know the value of k. In fact, the outline works perfectly fine if instead we know an estimation of k that is at least k and at most $k(1+\varepsilon/10)$ (Why? Explain what happens in the algorithm and how the calculations change). To remove the assumption of knowing this value, we simply run the algorithm for many estimates, which are powers of $(1+\varepsilon/10)$. That is, we run the algorithm for each estimate of the form $(1+\varepsilon/10)^i$ between 1 and n^2. Notice that there are only $O(\log_{1+\varepsilon/10} n)$ such powers, in this range. From each estimate, we get some cut (which might or might not be a small cut). The algorithm sets its output to be the smallest of these cuts. We know that one of these estimates $(1+\varepsilon/10)^i$ must be at least k and at most $k(1+\varepsilon/10)$. For that estimate, the algorithm outputs a cut that has size at most $(2+\varepsilon)k$. For other estimates, the cut size might be much larger, but since we output the minimum, we know that the output cut will have size at most $(2+\varepsilon)k$.

Chapter 3

Distributed Computing via All-to-All Communication

3.1 Introduction & the Congested Clique Model

In this chapter, we see a brief introduction to distributed computation in a setting where all the computers in the system can talk to each other (pairwise) via direct bounded-size message exchanges. This is sometimes also called the *congested clique* model of distributed computing.

The Congested Clique Model To model systems with all to all communication, we consider n processors that can communicate in synchronous rounds, in an all-to-all fashion. Per round, each processor can send $O(\log n)$ bits to each other processor—hence, $O(n \log n)$ bits in total. Notice that in one round, each processor can learn the unique identifier of all other processors (each processor sends its identifier to all other processors, directly). In fact, because of this, and as nodes can sort all identifiers locally once they known them, we can without loss of generality think that the nodes have unique identifiers from $\{1, 2, \dots, n\}$.

3.2 Routing in the Congested Clique

One of the most interesting problems, and also key building blocks in distributed algorithms in the congested clique model is the *routing* problem, stated as follows:

The Routing Problem: Suppose that there are a number of $O(\log n)$-bit messages, where the i^{th} message resides in some source node s_i and should be delivered to some target/destination node t_i. We emphasize that each node might the source and/or destination for several messages. Initially, for each message, only the source knows the related destination. The objective is to deliver each message from its source to the destination.

Intutive Discussion The interesting question is how many rounds of all-to-all communication do we need to solve this problem. Of course, the answer depends on the source and destinations. For instance, if each node wants to send exactly one message to each other node, that can be done directly in one round of the model with all-to-all communication. What else can we do? For instance, this solution doesn't work if one node wants to send several messages to some particular other node. What should be do then?

A concrete question is,

What instances of the routing problem can be solved in $O(1)$ rounds?

Can we characterize necessary and sufficient conditions for that?

A clear necessary condition is that each node should be the source for at most $O(n)$ messages, and each node should be at most the destination for at most $O(n)$ messages. This is because, per round, each node can send at most $n-1$ messages, one to each other node, and can receive at most $n-1$ messages, one from each other node. Interestingly, we see in this lecture that this necessary condition is also sufficient.

3.2.1 Viewing Routing as an Edge Coloring Problem

Let us think that we are in the setting that we know all the message source and destinations and we want to design a routing procedure, knowing all the information, in a centralized way. In the next subsection, we come back to the question of how to do such a thing in the distributed setting, when for each message, only the source of it knows which node is the destination.

We now argue that if each node is the source for at most K messages and each node is the destination for at most K messages, when $K \leqslant n$, the routing problem can be solved in $O(1)$ rounds. For that, we cast the routing problem as an instance of *edge coloring* for a certain graph.

Consider a bipartite graph H with n nodes on each side. That is, the graph is made of nodes $\{a_1, a_2, \dots, a_n\}$ on one side and nodes $\{b_1, b_2, \dots, b_n\}$. Now, draw an edge between a_i and b_j iff in the routing problem, there is a message that has source node i and destination j.

We can use edge colorings of this graph to solve the routing problem:

Lemma 3.1. *Any edge coloring of H with q colors implies a routing algorithm with $2\lceil q/n \rceil$ rounds.*

Proof. Consider a given coloring of H with q colors and partition the colors into $\lceil q/n \rceil$ parts, each of which has n colors. Let us focus on one part. We explain how the messages in the edges that are colored with this part of colors can be delivered in 2 rounds. Hence, over all the parts, all messages can be delivered in $2\lceil q/n \rceil$ rounds.

Focusing on one part of colors, let us renumber the at most n colors in this part so that they are from $[1, n]$. Consider all the edges (a_i, b_j) in the colors of this part, and the corresponding message that should go from node i of the system to node j. The key idea is this: we interpret the color of edge (a_i, b_j) as the identifier on an intermediate node. If the edge is colored with color $k \in [1, n]$, then, node i send the message destined to j instead to the intermediate node k and node k will then directly send the message to node j. Now, notice that this a correct procedure and it will never try to send two messages through the same edge in the same round. That is because, edges of H that have the same color k form a matching in H and thus, they are disjoint in sources and in destinations. Therefore, in the first round of communication, each source node sends at most one message to node k, and in the second round, node k should send at most one message to each node j. $\square$

Edge coloring of H: Recall that were are in the case that each node is the source for at most $K \leqslant n$ messages and each node is the destination for at most $K \leqslant n$ messages. Hence, the corresponding bipartite graph H has maximum degree at most n. By a theorem of Vizing, for any bipartite graph with maximum degree Δ, we can color its edges using Δ colors such that any two edges that share an endpoint have different colors[1]. Hence, there is a coloring of its edges with n colors. By Lemma 3.1, this implies we can solve the routing problem in 2 rounds.

3.2.2 Solving the Routing Problem Distributedly

The solution given above works if we know all of the source and destinations of all messages (and can solve the edge coloring problem in a centralized fashion). There is an elegant distributed algorithm by Christoph Lenzen [Len13][2] that solves the problem in $O(1)$ rounds, assuming each node is the source for at most K messages and each node is the destination for at most K messages, where $K \leqslant n$. Since describing this whole algorithm does not fit the time of one lecture, we instead describe a slightly weaker result. We show a randomized algorithm that solves the problem in $O(1)$ rounds, assuming $K \leqslant n/(20 \log n)$. We leave it as an (optional) exercise how to extend the algorithm to work when $K \leqslant O(n/\log\log n)$ and even further[3].

Distributed Routing for $K \leqslant n/(20 \log n)$ Make each source send its message to a $5 \log n$ independently chosen random intermediate node $k_1, \ldots, k_{5 \log n} \in [1, \ldots, n]$ and ask that intermediate node to deliver it directly to the destination. That is, for each message, we make $5 \log n$ copies of it and send toward the destination, through independently chosen random intermediate nodes. We use independent random intermediate points, for

[1]For general graphs (i.e., non-bipartite), Vizing's theorem implies a coloring with $\Delta + 1$ colors, and that is the best possible in general, e.g., think of a triangle graph.

[2]Who was a PhD student at ETH Zurich.

[3]One can apply the same idea repeatedly to get to $K \leqslant O(n/\log\log \ldots \log n)$, for any constant number of repetitions of log.

different messages. Also, the delivery process is done in two separate rounds: in the first round the message is sent from the source to the intermediate nodes, and in the second round, the message is sent from the intermediate nodes to the destination. If for an edge, there are 2 or more copies of messages that are planned to go through that edge in a round, we say that these copies failed.

Lemma 3.2. *With probability at least $1 - 1/n$, for each message, at least one of its copies arrives at the related destination.*

Proof. Let us focus on one message whose source is node i, and one fixed copy of it. What is the probability that this message fails to reach the intermediate node that it chooses? Notice that the copy fails in that step, only if chooses an edge $\{i, k\}$ that is also chosen by another copy of a message whose source is i. The only messages that can be arranged to go from i to k are messages whose source is i. There are at most $K \leqslant n/(20 \log n)$ such messages, and each such message has $5 \log n$ copies. Hence, at most $n/4$ edges starting from i are blocked with other copies of messages. Since the intermediate node k of the copy we are considering is chosen independent of everything else, we conclude that the probability of the copy failing in the first step is at most $1/4$.

You can see similarly that the probability of each copy failing in the second step — going from the intermediate node to the destination — is also at most $1/4$. That is because, for each destination j, there are at most $K \leqslant n/(20 \log n)$ messages destined to j, and each such message has $5 \log n$ copies. Hence, at most $n/4$ edge going to node j are blocked with other copies of messages.

By a union bound, we conclude that each copy of each message succeeds to reach its destination with probability at least $1 - (1/4 + 1/4) = 1/2$. Hence, considering that one message has $5 \log n$ copies, with probability at least $1 - (1/2)^{5 \log n} = 1 - 1/n^5$, at least one of the copies makes it to the destination. By a union bound over all the at most $Kn \leqslant n^2$ messages, we can conclude that with probability at least $1 - 1/n^3$, for each message, at least one of its copies makes it to the corresponding destination. $\square$

Exercise Extend the above method to solve the problem whenever $K \leqslant O(n/ \log \log n)$.

Hint: *think about having only $3 \log \log n$ copies per message, and then afterwards dealing with all the left over messages that none of their copies makes it to the destination.*

Bibliography

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

[ALGP89] Baruch Awerbuch, M Luby, AV Goldberg, and Serge A Plotkin. Network decomposition and locality in distributed computation. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 364–369. IEEE, 1989.

[AS04] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[Bar15] Leonid Barenboim. Deterministic (δ+ 1)-coloring in sublinear (in δ) time in static, dynamic and faulty networks. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 345–354. ACM, 2015.

[BFKL16] Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *arXiv preprint arXiv:1607.05127*, 2016.

[Bol78] Béla Bollobás. Chromatic number, girth and maximal degree. *Discrete Mathematics*, 24(3):311–314, 1978.

[CFS10] David Conlon, Jacob Fox, and Benny Sudakov. Hypergraph ramsey numbers. *Journal of the American Mathematical Society*, 23(1):247–266, 2010.

[CHGK14] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2014.

[CLRS01] Thomas H.. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001.

[CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 206–219. ACM, 1986.

[DEMN20] Michal Dory, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Distributed weighted min-cut in nearly-optimal time. *arXiv preprint arXiv:2004.09129*, 2020.

[DG19] Michal Dory and Mohsen Ghaffari. Improved distributed approximations for minimum-weight two-edge-connected spanning subgraph. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 521–530, 2019.

[DHIN19] Michael Dinitz, Magnús M Halldórsson, Taisuke Izumi, and Calvin Newport. Distributed minimum degree spanning trees. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 511–520, 2019.

[DHNS19] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proc. of the Symp. on Theory of Comp. (STOC)*, page arXiv:1904.04341, 2019.

[Dor18] Michal Dory. Distributed approximation of minimum k-edge-connected spanning subgraphs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 149–158, 2018.

[DSHK$^+$11] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 363–372, 2011.

[Elk04] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 331–340, 2004.

[Elk17] Michael Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 757–770, 2017.

[EN16] Michael Elkin and Ofer Neiman. On efficient distributed construction of near optimal routing schemes. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 235–244, 2016.

[Erd59] Paul Erdős. Graph theory and probability. *Canada J. Math*, 11:34G38, 1959.

[FHK16] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, 2016.

[FN18] Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 686–697. IEEE, 2018.

[GH16] Mohsen Ghaffari and Bernhard Haupler. Distributed algorithms for planar networks II: Low-congestion shortcuts, mst, and min-cut. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2016.

[GH20] Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. *arXiv preprint arXiv:2008.03091*, 2020.

[Gha14] Mohsen Ghaffari. Near-optimal distributed approximation of minimum-weight connected dominating set. In *the Pro. of the Int'l Colloquium on Automata, Languages and Programming (ICALP)*, 2014.

[GK13] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 1–15, 2013.

[GKK$^+$15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2015.

[GKS17] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distrbuted MST and routing in almost mixing time. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2017.

[GL14] Mohsen Ghaffari and Christoph Lenzen. Near-optimal distributed tree embedding. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, 2014.

[GL18a] Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 431–444, 2018.

[GL18b] Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[GPS87] Andrew Goldberg, Serge Plotkin, and Gregory Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 315–324. ACM, 1987.

[GTN20] Mohsen Ghaffari, Mikkel Thorup, and Krzysztof Nowicki. Faster algorithms for edge connectivity via random 2-out contractions. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2020.

[GU15] Mohsen Ghaffari and Rajan Udwani. Brief announcement: Distributed single-source reachability. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 163–165. ACM, 2015.

[HHW18] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. Round-and message-optimal distributed graph algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 119–128, 2018.

[HIZ16a] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016.

[HIZ16b] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *International Symposium on Distributed Computing (DISC)*, pages 158–172. Springer, 2016.

[HKN19] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. *SIAM Journal on Computing*, (0):STOC16–98, 2019.

[HLZ18] Bernhard Haeupler, Jason Li, and Goran Zuzic. Minor excluded network families admit fast distributed algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 465–474, 2018.

[Jam11] Mohammad Shoaib Jamall. A brooks' theorem for triangle-free graphs. *arXiv preprint arXiv:1106.1958*, 2011.

[Kim95] Jeong Han Kim. On brooks' theorem for sparse graphs. *Combinatorics, Probability and Computing*, 4(02):97–132, 1995.

[KP95] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 238–251, 1995.

[Kuh09] Fabian Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 138–144. ACM, 2009.

[KW06] Fabian Kuhn and Rogert Wattenhofer. On the complexity of distributed graph coloring. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 7–15. ACM, 2006.

[Len13] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 42–50, 2013.

[Lin87] Nathan Linial. Distributive graph algorithms global solutions from local data. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 331–335. IEEE, 1987.

[Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[LPS13] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: Extended abstract. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 381–390, 2013.

[LPS14] Christoph Lenzen and Boaz Patt-Shamir. Improved distributed steiner forest construction. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2014.

[LS91] Nathan Linial and Michael Saks. Decomposing graphs into regions of small diameter. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 320–330, 1991.

[LS14] Juhana Laurinharju and Jukka Suomela. Brief announcement: Linial's lower bound made easy. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 377–378. ACM, 2014.

[Lub66] David Lubell. A short proof of sperner's lemma. *Journal of Combinatorial Theory*, 1(2):299, 1966.

[Lub85] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 1–10, 1985.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. of the Symp. on Theory of Comp. (STOC)*, 2014.

[Nao91] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991.

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1):3–36, 2001.

[NO08] Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Foundations of Computer Science. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 327–336. IEEE, 2008.

[NS93] Moni Naor and Larry Stockmeyer. What can be computed locally? In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 184–193. ACM, 1993.

[NS95] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[NS14] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 439–453, 2014.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PR99] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 253–, 1999.

[PS92] Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 581–592. ACM, 1992.

[PS15] Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Information and Computation*, 243:263–280, 2015.

[RG20] Vaclav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *STOC*, page to appear, 2020.

[RS14] Joel Rybicki and Jukka Suomela. Exact bounds for distributed graph colouring. In *International Colloquium on Structural Information and Communication Complexity*, pages 46–60. Springer, 2014.

[Spe28] Emanuel Sperner. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.

[SV93] Márió Szegedy and Sundar Vishwanathan. Locality based graph coloring. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 201–207. ACM, 1993.