Juho Hirvonen and Jukka Suomela:

Distributed Algorithms 2020

https://jukkasuomela.fi/da2020/ · April 23, 2024

Chapter 4

LOCAL Model: Unique Identifiers

In the previous chapter, we studied deterministic distributed algorithms in port-numbered networks. In this chapter we will study a stronger model: *networks with unique identifiers*—see Figure 4.1. Following the standard terminology of the field, we will use the term "LOCAL model" to refer to networks with unique identifiers.

4.1 Definitions

Throughout this chapter, fix a constant $c > 1$. An assignment of *unique identifiers* for a port-numbered network $N = (V, P, p)$ is an injection

$$\mathrm{id}\colon V \to \{1, 2, \dots, |V|^c\}.$$

That is, each node $v \in V$ is labeled with a unique integer, and the labels are assumed to be relatively small.

Formally, unique identifiers can be interpreted as a graph problem Π', where each solution $\mathrm{id} \in \Pi'(N)$ is an assignment of unique identifiers for network N. If a distributed algorithm A solves a problem Π on a family $\mathscr{F}$ given Π', we say that *A solves Π on $\mathscr{F}$ given unique identifiers*, or equivalently, *A solves Π on $\mathscr{F}$ in the* LOCAL *model*.

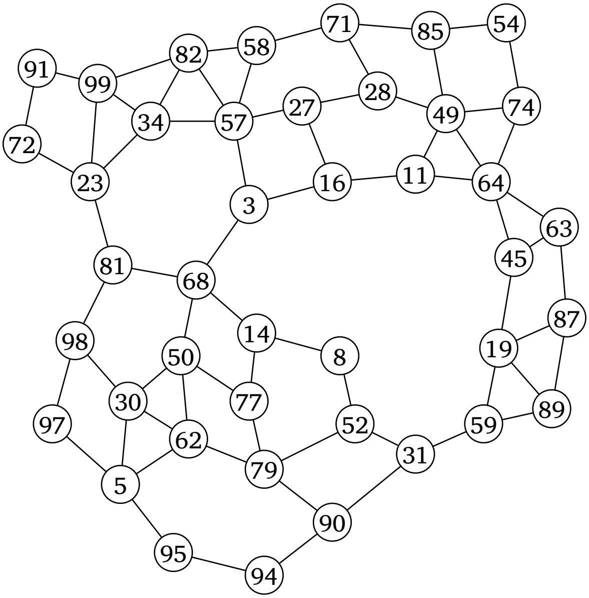

Figure 4.1: A network with unique identifiers.

For the sake of convenience, when we discuss networks with unique identifiers, we will identify a node with its unique identifier, i.e., $v = \mathrm{id}(v)$ for all $v \in V$.

4.2 Gathering Everything

In the LOCAL model, if the underlying graph $G = (V, E)$ is connected, all nodes can learn everything about G in time $O(\mathrm{diam}(G))$. In this section, we will present a gathering algorithm that accomplishes this.

In the gathering algorithm, each node $v \in V$ will construct sets $V(v, r)$ and $E(v, r)$, where $r = 1, 2, \dots$. For all $v \in V$ and $r \geq 1$, these sets will satisfy

$$V(v,r) = \mathrm{ball}_G(v,r), \tag{4.1}$$

$$E(v,r) = \big\{\{s,t\} : s \in \mathrm{ball}_G(v,r),\ t \in \mathrm{ball}_G(v,r-1)\big\}. \tag{4.2}$$

Now define the graph

$$G(v,r) = (V(v,r), E(v,r)). \tag{4.3}$$

See Figure 4.2 for an illustration.

The following properties are straightforward corollaries of (4.1)–(4.3).

(a) Graph $G(v,r)$ is a subgraph of $G(v,r+1)$, which is a subgraph of G.

(b) If G is a connected graph, and $r \geq \mathrm{diam}(G) + 1$, we have $G(v,r) = G$.

(c) If G_v is the connected component of G that contains v, and $r \geq \mathrm{diam}(G_v) + 1$, we have $G(v,r) = G_v$.

(d) For a sufficiently large r, we have $G(v,r) = G(v,r+1)$.

(e) If $G(v,r) = G(v,r+1)$, we will also have $G(v,r+1) = G(v,r+2)$.

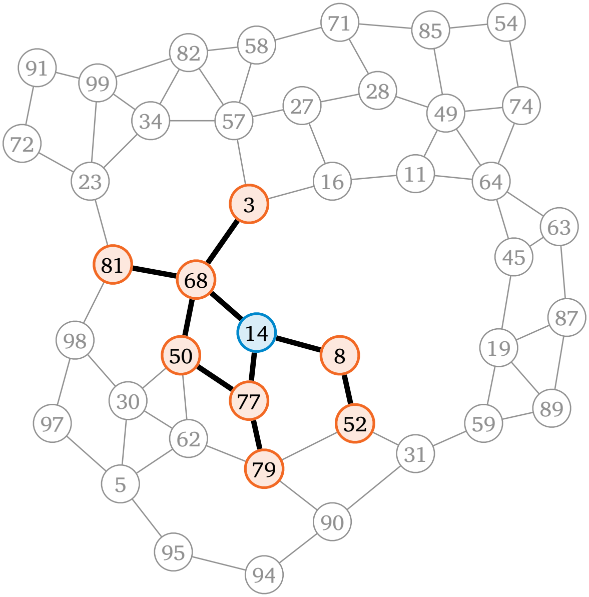

Figure 4.2: Subgraph $G(v, r)$ defined in (4.3), for $v = 14$ and $r = 2$.

(f) Graph $G(v,r)$ for $r > 1$ can be constructed recursively as follows:

$$V(v,r) = \bigcup_{u \in V(v,1)} V(u,r-1), \tag{4.4}$$

$$E(v,r) = \bigcup_{u \in V(v,1)} E(u,r-1). \tag{4.5}$$

The gathering algorithm maintains the following invariant: after round $r \geq 1$, each node $v \in V$ has constructed graph $G(v,r)$. The execution of the algorithm proceeds as follows:

(a) In round 1, each node $u \in V$ sends its identity u to each of its ports. Hence after round 1, each node $v \in V$ knows its own identity and the identities of its neighbors. Put otherwise, v knows precisely $G(v,1)$.

(b) In round $r > 1$, each node $u \in V$ sends $G(u,r-1)$ to each of its ports. Hence after round r, each node $v \in V$ knows $G(u,r-1)$ for all $u \in V(v,1)$. Now v can reconstruct $G(v,r)$ using (4.4) and (4.5).

(c) A node $v \in V$ can stop once it detects that the graph $G(v,r)$ no longer changes.

It is easy to extend the gathering algorithm so that we can discover not only the underlying graph $G = (V,E)$ but also the original port-numbered network $N = (V,P,p)$.

4.3 Solving Everything

Let $\mathscr{F}$ be a family of connected graphs, and let Π be a distributed graph problem. Assume that there is a deterministic *centralized* (non-distributed) algorithm A' that solves Π on $\mathscr{F}$. For example, A' can be a simple brute-force algorithm—we are not interested in the running time of algorithm A'.

Now there is a simple distributed algorithm A that solves Π on $\mathscr{F}$ in the LOCAL model. Let $N = (V, P, p)$ be a port-numbered network with the underlying graph $G \in \mathscr{F}$. Algorithm A proceeds as follows.

(a) All nodes discover N using the gathering algorithm from Section 4.2.

(b) All nodes use the centralized algorithm A' to find a solution $f \in \Pi(N)$. From the perspective of algorithm A, this is merely a state transition; it is a local step that requires no communication at all, and hence takes 0 communication rounds.

(c) Finally, each node $v \in V$ switches to state $f(v)$ and stops.

Clearly, the running time of the algorithm is $O(\operatorname{diam}(G))$.

It is essential that all nodes have the same canonical representation of network N (for example, V, P, and p are represented as lists that are ordered lexicographically by node identifiers and port numbers), and that all nodes use the same deterministic algorithm A' to solve Π. This way we are guaranteed that all nodes have locally computed the *same* solution f, and hence the outputs $f(v)$ are globally consistent.

4.4 Focus on Computational Complexity

So far we have learned the key difference between PN and LOCAL models: while there are plenty of graph problems that cannot be solved at all in the PN model, we know that all computable graph problems can be easily solved in the LOCAL model.

Hence our focus shifts from computability to computational complexity. While it is trivial to determine if a problem can be solved in the LOCAL model, we would like to know which problems can be solved quickly. In particular, we would like to learn which problems can be solved in time that is much smaller than $\operatorname{diam}(G)$. It turns out that graph coloring is an example of such a problem.

In the rest of this chapter, we will design an efficient distributed algorithm that finds a graph coloring in the LOCAL model. The algorithm will find a proper vertex coloring with $\Delta + 1$ colors in $O(\Delta + \log^* n)$ communication rounds, for any graph with $n = |V|$ nodes and maximum degree Δ. We will start with a simple greedy algorithm that we will later use as a subroutine.

4.5 Greedy Color Reduction

Let $x \in \mathbb{N}$. We present a greedy color reduction algorithm that reduces the number of colors from x to

$$y = \max\{x - 1, \Delta + 1\},$$

where Δ is the maximum degree of the graph. That is, given a proper vertex coloring with x colors, the algorithm outputs a proper vertex coloring with y colors. The running time of the algorithm is one communication round.

4.5.1 Algorithm

The algorithm proceeds as follows; here f is the x-coloring that we are given as input and g is the y-coloring that we produce as output. See Figure 4.3 for an illustration.

(a) In the first communication round, each node $v \in V$ sends its color $f(v)$ to each of its neighbors.

(b) Now each node $v \in V$ knows the set

$$C(v) = \{i : \text{there is a neighbor } u \text{ of } v \text{ with } f(u) = i\}.$$

We say that a node is *active* if $f(v) > \max C(v)$; otherwise it is *passive*. That is, the colors of the active nodes are local maxima. Let

$$\bar{C}(v) = \{1, 2, \dots\} \setminus C(v)$$

be the set of *free colors* in the neighborhood of v.

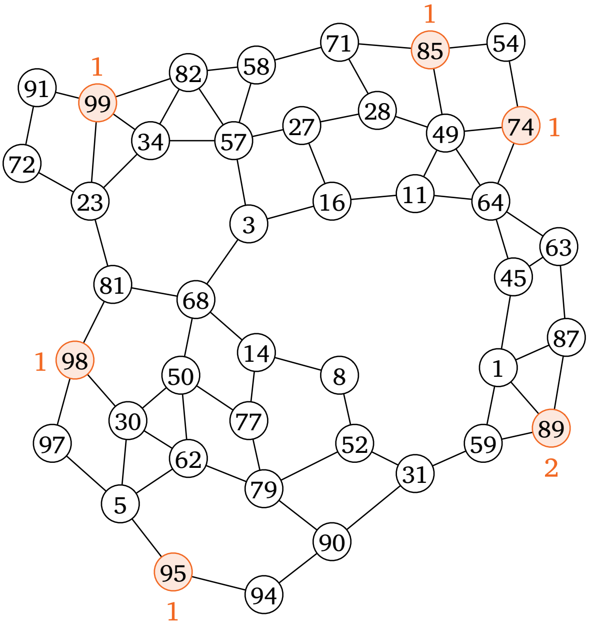

Figure 4.3: Greedy color reduction. The active nodes have been highlighted. Note that in the original coloring f, the largest color was 99, while in the new coloring, the largest color is strictly smaller than 99—we have successfully reduced the number of colors in the graph.

(c) A node $v \in V$ outputs

$$g(v) = \begin{cases} f(v) & \text{if } v \text{ is passive,} \\ \min \bar{C}(v) & \text{if } v \text{ is active.} \end{cases}$$

Informally, a node whose color is a local maximum re-colors itself with the first available free color.

4.5.2 Analysis

Lemma 4.1. *The greedy color reduction algorithm reduces the number of colors from x to*

$$y = \max\{x-1, \Delta+1\},$$

where Δ is the maximum degree of the graph.

Proof. Let us first prove that $g(v) \in \{1, 2, \ldots, y\}$ for all $v \in V$. As f is a proper coloring, we cannot have $f(v) = \max C(v)$. Hence there are only two possibilities.

(a) $f(v) < \max C(v)$. Now v is passive, and it is adjacent to a node u such that $f(v) < f(u)$. We have

$$g(v) = f(v) \leq f(u) - 1 \leq x - 1 \leq y.$$

(b) $f(v) > \max C(v)$. Now v is active, and we have

$$g(v) = \min \bar{C}(v).$$

There is at least one value $i \in \{1, 2, \ldots, |C(v)| + 1\}$ with $i \notin C(v)$; hence

$$\min \bar{C}(v) \leq |C(v)| + 1 \leq \deg_G(v) + 1 \leq \Delta + 1 \leq y.$$

Next we will show that g is a proper vertex coloring of G. Let $\{u, v\} \in E$. If both u and v are passive, we have

$$g(u) = f(u) \neq f(v) = g(v).$$

Otherwise, w.l.o.g., assume that u is active. Then we must have $f(u) > f(v)$. It follows that $f(u) \in C(v)$ and $f(v) \leq \max C(v)$; therefore v is passive. Now $g(u) \notin C(u)$ while $g(v) = f(v) \in C(u)$; we have $g(u) \neq g(v)$. □

The key observation is that the set of active nodes forms an independent set. Therefore all active nodes can pick their new colors simultaneously in parallel, without any risk of choosing colors that might conflict with each other.

4.5.3 Remarks

The greedy color reduction algorithm does not need to know the number of colors x or the maximum degree Δ; we only used them in the analysis. We can take any graph, blindly apply greedy color reduction, and we are guaranteed to reduce the number of colors by one—provided that the number of colors was larger than $\Delta + 1$. In particular, we can apply the greedy color reduction repeatedly until we get stuck, at which point we have a $(\Delta + 1)$-coloring of G—we will formalize and generalize this idea in Exercise 4.3.

4.6 Efficient $(\Delta + 1)$-coloring

In the remaining sections we will describe two coloring algorithms that, together with the greedy algorithm from the previous section, can be used to $(\Delta + 1)$-color graphs of maximum degree Δ.

On a high level, the $(\Delta + 1)$-coloring algorithm is composed of the following subroutines:

(a) Algorithm from Section 4.8: Using unique identifiers as input, compute an $O(\Delta^2)$-coloring x in $O(\log^* n)$ rounds.

(b) Algorithm from Section 4.7: Given x as input, compute an $O(\Delta)$-coloring y in $O(\Delta)$ rounds.

(c) Algorithm from Section 4.5: Given y as input, compute a $(\Delta + 1)$-coloring z in $O(\Delta)$ rounds.

We have already seen the greedy algorithm that we will use in the final step; we will proceed in the reverse order and present next the algorithm that turns an $O(\Delta^2)$-coloring into an $O(\Delta)$-coloring. In what follows, we will assume that the nodes are given the values of Δ and n as input; these assumptions will simplify the algorithms significantly.

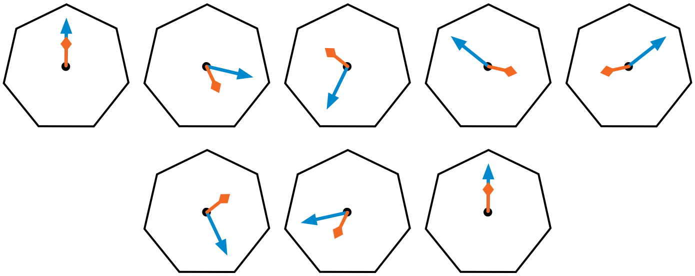

Figure 4.4: Two clocks for $q = 7$. The blue hand moves 2 steps per time unit, and the orange hand 3 steps. Hands moving at different speeds meet again after q moves, but not before.

4.7 Additive-Group Coloring

Consider two clocks with q steps, for any prime q; see Figure 4.4. The first clock moves its hand a steps in each time unit, and the second clock moves its hand $b \neq a$ steps in each time unit. Starting from the same position, when are the two hands in the same position again?

It is a fundamental property of *finite fields* that they are in the same position again after exactly q steps. We recap definitions and facts about finite fields in Section 4.15.

Building on this observation, we construct an algorithm where each node behaves like a clock with one hand, turning its hand with some constant speed. We will use the input coloring to ensure that *clocks with the same starting position turn their hands at different speeds*. Then we will simply wait until a clock is in a position not shared by any of the neighbors, and this position becomes the final color of the node. If we do not have too many neighbors, each node will eventually find such a position, leading to a proper coloring.

4.7.1 Algorithm

Let q be a prime number with $q > 2\Delta$. We assume that we are given a coloring with q^2 colors, and we will show how to construct a coloring with q colors in $O(q)$ rounds. Put otherwise, we can turn a coloring with $O(\Delta^2)$ colors into a coloring with $O(\Delta)$ colors in $O(\Delta)$ rounds, as long as we choose our prime number q in a suitable manner.

If we have an input coloring with q^2 colors, we can represent the color of node v as a pair $f(v) = \langle f_1(v), f_2(v)\rangle$ where $f_1(v)$ and $f_2(v)$ are integers between 0 and $q-1$.

Using the clock analogue, v can be seen as a clock with the hand at position $f_2(v)$, turning at speed $f_1(v)$. In the algorithm we will stop clocks by setting $f_1(v) = 0$ whenever this is possible in a conflict-free manner. When all clocks have stopped, all nodes have colors of the form $\langle 0, f_2(v)\rangle$ where $f_2(v)$ is between 0 and $q-1$, and hence we have got a proper coloring with q colors.

We say that two colors $\langle a_1, a_2\rangle$ and $\langle b_1, b_2\rangle$ are in *conflict* if $a_2 = b_2$. The algorithm repeatedly performs the following steps:

- Each node sends its current colors to each neighbor.
- For each node v, if $f(v)$ is in conflict with any neighbor, set

  $$f(v) \leftarrow \big\langle f_1(v), (f_1(v) + f_2(v)) \bmod q \big\rangle.$$

  Otherwise, set

  $$f(v) \leftarrow \big\langle 0, f_2(v) \big\rangle.$$

In essence, we stop non-conflicting clocks and keep moving all other clocks at a constant rate. We say that a node v is *stopped* when $f_1(v) = 0$; otherwise it is *running*; note a stopped node will not change its color any more.

We show that after $O(q)$ iterations of this loop, all nodes will be stopped, and they form a proper coloring—assuming we started with a proper coloring.

4.7.2 Correctness

First, we show that in each iteration a proper coloring remains proper. In what follows, we use f to denote the coloring before one iteration and

g to denote the coloring after the iteration. Consider a fixed node v and an arbitrary neighbor u. We show by a case analysis that $f(v) \neq f(u)$ implies $g(v) \neq g(u)$.

(1) Assume that v is stopped after this round; then $g(v) = \langle 0, f_2(v) \rangle$.

   (a) If $f_1(u) = 0$, then u has stopped and $g(u) = f(u)$. By assumption $f(v) \neq f(u)$ and therefore $g(v) \neq g(u)$.

   (b) If $f_1(u) \neq 0$ and $f(u)$ is not in conflict with its neighbors, then $g(u) = \langle 0, f_2(u) \rangle$. As there are no conflicts with v, we must have $f_2(v) \neq f_2(u)$, and therefore $g(v) \neq g(u)$.

   (c) Otherwise $f_1(u) \neq 0$ and $f(u)$ is in conflict with a neighboring color. Then $g_1(u) = f_1(u) \neq 0 = g_1(v)$, and therefore $g(v) \neq g(u)$.

(2) Otherwise we have $g(v) = \langle f_1(v), (f_1(v) + f_2(v)) \bmod q \rangle$, where $f_1(v) \neq 0$.

   (a) If u has stopped, then $g_1(u) = 0$, and therefore $g(v) \neq g(u)$.

   (b) Otherwise u is running. Then

$$g(u) = \langle f_1(u), (f_1(u) + f_2(u)) \bmod q \rangle.$$

   If $f_1(v) \neq f_1(u)$, we will have $g_1(v) \neq g_1(u)$ and therefore $g(v) \neq g(u)$. Otherwise $f_1(v) = f_1(u)$ but then by assumption we must have $f_2(v) \neq f_2(u)$, which implies $g_2(v) \neq g_2(u)$ and therefore $g(v) \neq g(u)$.

4.7.3 Running Time

Next we analyze the running time. Assume that we start with a proper coloring f. We want to show that after a sufficient number of iterations of the additive-group algorithm, each node must have had an iteration in which its color did not conflict with color of its neighbors, and hence got an opportunity to stop.

Let f^0 denote the initial coloring before the first iteration and let f^i denote the coloring after iteration $i = 1, 2, \ldots$. The following lemma shows that two running nodes do not conflict too often during the execution.

Lemma 4.2. *Consider t consecutive iterations of the additive-group coloring algorithm, for $t \leq q$. Let u and v be adjacent nodes such that both of them are still running before iteration t. Then there is at most one iteration $i = 0, 1, \ldots, t-1$ with a conflict $f_2^i(u) = f_2^i(v)$.*

Proof. Assume that for some i we have $f^i(u) = \langle a, b \rangle$ and $f^i(v) = \langle a', b \rangle$ with $a \neq a'$. In the subsequent iterations $j = i+1, i+2, \ldots$, we have

$$f_2^j(u) - f_2^j(v) \equiv (a - a')(j - i) \mod q.$$

Assume that for some j we have another conflict $f_2^j(u) = f_2^j(v)$, implying that $(a - a')(j - i) \equiv 0 \mod q$. If a prime divides a product xy of two integers x and y, then it also divides x or y (Euclid's Lemma). But $a - a'$ cannot be a multiple of q, since $a \neq a'$ and $0 \leq a, a' < q$, and $j - i$ cannot be a multiple of q, either, since $0 \leq i < j < q$. □

We also need to show that a node is not in conflict with a stopped node too often.

Lemma 4.3. *Consider t consecutive iterations of the additive-group coloring algorithm, for $t \leq q$. Let u and v be adjacent nodes such that u is still running before iteration t but v was stopped before iteration* 1. *Then there is at most one iteration $i = 0, 1, \ldots, t-1$ with a conflict $f_2^i(u) = f_2^i(v)$.*

Proof. The same argument as in the proof of Lemma 4.2 works, this time with $a' = 0$. □

It remains to show that, based on Lemmas 4.2 and 4.3, the algorithm finishes fast.

Consider a sequence of consecutive $q > 2\Delta$ iterations of the additive-group coloring algorithm starting with any initial coloring f. Consider an arbitrary node u that does not stop during any of these rounds. Let v

be a neighbor of u. No matter if and when v stops, the color of v will conflict with color of u at most twice during the q rounds:

- Consider the rounds (if any) in which v is running. There are at most q such rounds. By Lemma 4.2, u conflicts with v at most once during these rounds.

- Consider the remaining rounds (if any) in which v is stopped. There are at most q such rounds. By Lemma 4.3, u conflicts with v at most once during these rounds.

So for each neighbor v of u, there are at most 2 rounds among q rounds such that the color of v conflicts with the color of u. As there are at most Δ neighbors, there are at most 2Δ rounds among q rounds such that the color of some neighbor of u conflicts with the current color of u. But $q > 2\Delta$, so there has to be at least one round after which none of the neighbors are in conflict with u—and hence there will be an opportunity for u to stop.

4.8 Fast $O(\Delta^2)$-coloring

The additive-group coloring algorithm assumes that we start with an $O(\Delta^2)$-coloring of the network. In this section we present an algorithm that computes an $O(\Delta^2)$-coloring in $O(\log^* n)$ communication rounds.

The algorithm proceeds in two phases. In the first phase, the coloring given by the unique identifiers is iteratively reduced to an $O(\Delta^2 \log^2 \Delta)$-coloring. In the second phase, a final color reduction step yields an $O(\Delta^2)$ coloring.

Both phases are based on the same combinatorial construction, called a cover-free set family. We begin by describing the construction for the first phase.

4.8.1 Cover-Free Set Families

The coloring algorithm is based on the existence of non-covering families of sets. Intuitively, these are families of sets such that any two sets do

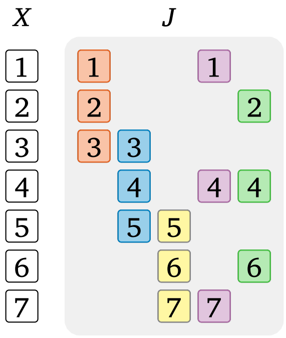

Figure 4.5: A 2-cover-free set family J of 5 subsets of a base set X on 7 elements. No two sets cover a third distinct set.

not have a large overlap: then no small collection of sets contains all elements in another set. Therefore, if each node is assigned such a set, it can find an element that is not in the sets of its neighbors, and pick that element as its new color.

A family J of n subsets of $\{1, \ldots, m\}$ is *k-cover-free* if for every $S \in J$ and every collection of k sets $S_1, \ldots, S_k \in J$ distinct from S we have that

$$S \nsubseteq \bigcup_{i=1}^{k} S_i .$$

See Figure 4.5 for an example.

4.8.2 Constructing Cover-Free Set Families

Cover-free set families can be constructed using *polynomials over finite fields*. The example of finite fields we are interested in is GF(q) for a prime q, which is simply modular arithmetic of integers modulo q. We consider polynomials over such a field. A brief recap is given in Section 4.15.

A basic result about polynomials states that two distinct polynomials evaluate to the same value at a bounded number of points

Lemma 4.4. *Let f, g be two distinct polynomials of degree d over a finite field* $\mathrm{GF}(q)$*, for some prime q. Then $f(x) = g(x)$ holds for at most d elements $x \in \mathrm{GF}(q)$.*

Proof. See Section 4.15. □

Now fix a prime q. Our base set will be $X = \mathrm{GF}(q) \times \mathrm{GF}(q)$. Thus we have that $|X| = m = q^2$.

For a positive natural number d, consider $\mathrm{Poly}(d, q)$, the set of polynomials of degree d over $\mathrm{GF}(q)$. For each polynomial $g \in \mathrm{Poly}(d, q)$, fix the set

$$S_g = \big\{(a, g(a)) \bigm| a \in \mathrm{GF}(q)\big\}$$

that is associated with this polynomial. Note that each S_g contains exactly q elements: one for each element of $\mathrm{GF}(q)$. Then we can define the family

$$J = J_{d,q} = \big\{S_g \bigm| g \in \mathrm{Poly}(d, q)\big\}.$$

Consider any two distinct polynomials f and g in $\mathrm{Poly}(d, q)$: by Lemma 4.4 there are at most d elements a such that $f(a) = g(a)$. Therefore $|S_f \cap S_g| \le d$, and J is a $\lfloor q/d \rfloor$-cover-free set family.

Any polynomial is uniquely defined by its coefficients. Therefore the set $J_{d,q}$ has size q^{d+1}, as it consists of a set of pairs for each polynomial of degree d.

By choosing parameters q and d we can construct a Δ-cover free family that can be used to color efficiently.

Lemma 4.5. *For all integers x, Δ such that $x > \Delta \ge 2$, there exists a Δ-cover-free family J of x subsets from a base set of $m \le 4(\Delta + 1)^2 \log^2 x$ elements.*

Proof. We begin by choosing a prime q such that

$$\big\lfloor (\Delta + 1) \log x \big\rfloor \le q \le 2 \cdot \big\lfloor (\Delta + 1) \log x \big\rfloor.$$

By the Bertrand–Chebyshev theorem such a prime must always exist. Set $d = \lfloor \log x \rfloor$. By the previous observation, the family $J_{d,q}$, for the above parameter settings, is a $\lfloor q/d \rfloor$-cover-free family, where

$$\lfloor q/d \rfloor \geq \left\lfloor \frac{\lfloor (\Delta+1)\log x \rfloor}{\lfloor \log x \rfloor} \right\rfloor \geq \left\lfloor \frac{(\Delta+1)\log x - 1}{\log x} \right\rfloor \geq \Delta.$$

There are at least

$$q^{d+1} \geq (\Delta \log x)^{\log x} > x$$

sets in $J_{d,q}$, so we can choose x of them. The base set has

$$q^2 \leq 4(\Delta+1)^2 \log^2 x$$

elements. □

4.8.3 Efficient Color Reduction

Using Δ-cover-free sets we can construct an algorithm that reduces the number of colors from x to $y \leq 4(\Delta+1)^2 \log^2 x$ in one communication round, as long as $x > \Delta$.

Let f denote the input x-coloring and g the output y-coloring. Assume that J is a Δ-cover-free family of x sets on a base set of y elements, as in Lemma 4.5, that is ordered as $S_1, S_2, \ldots, S_x$. The algorithm functions as follows.

(a) Each node $v \in V$ sends its current color $f(v)$ to each of its neighbors.

(b) Each node receives the colors $f(u)$ of its neighbors $u \in N(v)$. Then it constructs the set $S_{f(v)}$, and the sets $S_{f(u)}$ for all $u \in N(v)$. Since $f(v) \neq f(u)$ for all $u \in N(v)$, and J is a Δ-cover-free family, we have that

$$S_{f(v)} \nsubseteq \bigcup_{u \in N(v)} S_{f(u)}.$$

In particular, there exists at least one $c \in S_{f(v)} \setminus \cup_{u \in N(v)} S_{f(u)}$. Node v sets $g(v) = c$ for the smallest such c.

Now assume that f is a proper coloring, that is, $f(v) \neq f(u)$ for all neighbors v and u. This implies that for each node v, each of its neighbors u selects a set that is different from $S_{f(v)}$; overall, the neighbors will select at most Δ distinct sets. Since J is a Δ-cover-free family, each node v can find an element $c \in S_{f(v)}$ that is not in the sets of its neighbors. Therefore setting $g(v) = c$ forms a proper coloring. Finally, since the sets $S \in J$ are subsets of $\{1, \dots, y\}$, for $y \leq 4(\Delta+1)^2(\log x)^2$, we have that g is a y-coloring.

4.8.4 Iterated Color Reduction

By a repeated application of the color reduction algorithm it is possible to reduce the number of colors down to $O(\Delta^2 \log^2 \Delta)$. Assuming we start with an input x-coloring, this will take $O(\log^* x)$ rounds.

We will now show that $O(\log^* x)$ iterations of the color reduction algorithm will reduce the number of colors from x to $O(\Delta^2 \log^2 \Delta)$. We assume that in the beginning, both x and Δ are known. Therefore after each iteration, all nodes know the total number of colors.

Assume that $x > 4(\Delta+1)^2 \log^2 \Delta$. Repeated iterations of the color reduction algorithm reduce the number of colors as follows:

$$\begin{aligned} x_0 &\mapsto x_1 \leq 4(\Delta+1)^2 \log^2 x, \\ x_1 &\mapsto x_2 \leq 4(\Delta+1)^2 \log^2(4(\Delta+1)^2 \log^2 x) \\ &\qquad = 4(\Delta+1)^2 \bigl(\log 4 + 2\log(\Delta+1) + 2\log\log x\bigr)^2. \end{aligned}$$

If $\log\log x \geq \log 4 + 2\log(\Delta+1)$, we have that

$$\begin{aligned} x_2 &\leq 4(\Delta+1)^2 (3 \log\log x)^2 \\ &= \bigl(6(\Delta+1)\log\log x\bigr)^2. \end{aligned}$$

In the next step, we reduce colors as follows:

$$\begin{aligned} x_2 &\mapsto x_3 \leq 4(\Delta+1)^2 \log^2\bigl(36(\Delta+1)^2(\log\log x)^2\bigr) \\ &\qquad = 4(\Delta+1)^2 \bigl(\log 36 + 2\log(\Delta+1) + 2\log\log\log x\bigr)^2. \end{aligned}$$

If $\log\log\log x \geq \log 36 + 2\log(\Delta+1)$, we have that

$$\begin{aligned} x_3 &\leq 4(\Delta+1)^2(3\log\log\log x)^2 \\ &= \big(6(\Delta+1)\log\log\log x\big)^2. \end{aligned}$$

Now we can see the pattern: as long as

$$\log^{(i)} x \geq \log 36 + 2\log(\Delta+1),$$

where $\log^{(i)} x$ is the i times iterated logarithm of x, we reduce colors from $(6(\Delta+1)\log^{(i-1)} x)^2$ to $(6(\Delta+1)\log^{(i)} x)^2$ in the ith step.

Once $\log^{(i)} x \geq \log 36 + 2\log(\Delta+1)$ no longer holds, we have a coloring with at most

$$c_\Delta = 4(\Delta+1)^2\big(3(\log 36 + 2\log(\Delta+1))\big)^2$$

colors. We can numerically verify that for all $\Delta \geq 2$, we have that

$$4(\Delta+1)^2\big(3(\log 36 + 2\log(\Delta+1))\big)^2 \leq (11(\Delta+1))^3.$$

We will use this observation in the next step.

It remains to calculate how many color reduction steps are required. By definition, after $T = \log^* x$ iterations we have that $\log^{(T)} x \leq 1$. Thus, after at most $\log^* x$ iterations of the color reduction algorithm we have a coloring with at most c_Δ colors.

4.8.5 Final Color Reduction Step

In the last step, we will reduce the coloring to an $O(\Delta^2)$-coloring. We will use another construction of Δ-cover-free families based on polynomials.

Lemma 4.6. *For all Δ, there exists a Δ-cover-free family J of x subsets from a base set of $m \leq (22(\Delta+1))^2$ elements for $x \leq (11(\Delta+1))^3$.*

This immediately gives us the following color reduction algorithm.

Corollary 4.7. *There is a distributed algorithm that, given a* $(11(\Delta+1))^3$*-coloring as an input, in one round computes a* $(22(\Delta+1))^2$*-coloring.*

Proof of Lemma 4.6. Our base set will be X with $|X| = m = q^2$, for a prime q. Again it is useful to see $X = \mathrm{GF}(q) \times \mathrm{GF}(q)$ as pairs of elements from the finite field over q elements.

Now consider polynomials $\mathrm{Poly}(2, q)$ of degree 2 over $\mathrm{GF}(q)$. For each such polynomial $g \in \mathrm{Poly}(2, q)$, let

$$S_g = \left\{(a, g(a)) \;\middle|\; a \in \mathrm{GF}(q)\right\}$$

be the pairs defined by the valuations of the polynomial g at each element of $\mathrm{GF}(q)$. We have that $|S_g| = q$ for all g.

Now we can construct the family

$$J = J_{2,q} = \left\{S_g \;\middle|\; g \in \mathrm{Poly}(2, q)\right\}$$

as the collection of point sets defined by all polynomials of degree 2. We have that $|J| = q^3$ since a polynomial is uniquely determined by its coefficients.

By Lemma 4.4, we have that $|S_f \cap S_g| \leq 2$ for any distinct polynomials $f, g \in P(2, q)$. Therefore covering any set S_g requires at least $\lceil q/2 \rceil$ other sets (distinct from S_g) from J.

We are now ready to prove that J is a Δ-cover-free family for suitable parameter settings. Since each set S_g contains q elements, and the intersection between the sets of distinct polynomials is at most 2, we want to find q such that $2\Delta \leq q - 1$ and q^3 is large enough. Using the Bertrand–Chebyshev theorem we know that there exists a prime q such that

$$11(\Delta+1) \leq q \leq 22(\Delta+1).$$

Any value q from this range is large enough. The base set X has size

$$m = q^2 \leq (22(\Delta+1))^2.$$

The family J has size

$$|J| \geq (11(\Delta+1))^3.$$

Finally, since we choose $q \geq 2\Delta + 1$, we have that no collection of Δ sets $\mathscr{S} = \{S_1, S_2, \ldots, S_\Delta\} \subseteq J$ can cover a set $S \notin \mathscr{S}$. □

4.9 Putting Things Together

It remains to show how to use the three algorithms we have seen so far together.

Theorem 4.8. *Assume that we know parameters Δ and n, and some polynomial bound n^c on the size of the unique identifiers. Graphs on n vertices with maximum degree Δ can be $(\Delta+1)$-colored in $O(\Delta+\log^* n)$ rounds in the* LOCAL *model.*

Proof. We begin with the unique identifiers, and treat them as an initial coloring with n^c colors.

(a) In the first phase we run the efficient color reduction algorithm from Section 4.8.3 for $T_1 = \log^*(n^c) = O(\log^* n)$ rounds to produce a coloring y_1 with at most $(11(\Delta+1))^3$ colors.

(b) In the second phase, after T_1 rounds have passed, each vertex can apply the final color reduction step from Section 4.8.5 to compute a coloring y_2. This reduces colors from $(11(\Delta+1))^3$ to $(22(\Delta+1))^2$.

(c) After T_1+1 rounds, we have computed an $O(\Delta^2)$-coloring y_2. Now each vertex runs the additive-group coloring algorithm from Section 4.7, applying it with y_2 as input. For a parameter $q \le 2\sqrt{(22(\Delta+1))^2} = 44\Delta+44$, this algorithm runs for $T_2 = q$ steps and computes a q-coloring y_3.

(d) In the last phase, after T_1+1+T_2 rounds, we apply the greedy color reduction algorithm from Section 4.5 iteratively $T_3 = 43\Delta+43$ times. Each iteration requires one round and reduces the maximum color by one.

After a total of

$$\begin{aligned} T_1+1+T_2+T_3 &\le \log^*(n^c)+87\Delta+88 \\ &= O(\Delta+\log^* n) \end{aligned}$$

rounds, we have computed a $(\Delta+1)$-coloring. □

4.10 Quiz

Consider the algorithm from Section 4.7.1 in the following setting:

- The network is a complete graph with $n = 4$ nodes; hence the maximum degree is $\Delta = 3$, and we can choose $q = 7 > 2\Delta$.
- We are given a coloring with $q^2 = 49$ colors; we can represent the possible input colors as pairs $(0,0), (0,1), \dots, (6,6)$.

Give an example of an input coloring such that we need to do exactly 6 iterations of the algorithm until all nodes have reached their final colors, i.e., colors of the form $(0, x)$.

Please give the answer by listing the four original color pairs of the nodes in any order; for example, if we asked for a coloring in which you need exactly 3 iterations, this would be a correct answer: $(2,3)$, $(3,2)$, $(3,6)$, $(4,6)$.

4.11 Exercises

Exercise 4.1 (applications). Let Δ be a known constant, and let $\mathscr{F}$ be the family of graphs of maximum degree at most Δ. Design fast distributed algorithms that solve the following problems on $\mathscr{F}$ in the LOCAL model.

(a) Maximal independent set.

(b) Maximal matching.

(c) Edge coloring with $O(\Delta)$ colors.

You can assume that all nodes get the value of n (the number of nodes) as input; also the parameter c in the identifier space is a known constant, so all nodes know the range of unique identifiers.

Exercise 4.2 (vertex cover). Let $\mathscr{F}$ consist of cycle graphs. Design a fast distributed algorithm that finds a 1.1-approximation of a minimum vertex cover on $\mathscr{F}$ in the LOCAL model.

▷ *hint A*

Exercise 4.3 (iterated greedy). Design a color reduction algorithm A with the following properties: given any graph $G = (V, E)$ and any proper vertex coloring f, algorithm A outputs a proper vertex coloring g such that for each node $v \in V$ we have $g(v) \le \deg_G(v) + 1$.

Let Δ be the maximum degree of G, let $n = |V|$ be the number of nodes in G, and let x be the number of colors in coloring f. The running time of A should be at most

$$\min\{n, x\} + O(1).$$

Note that the algorithm does not know n, x, or Δ. Also note that we may have either $x \le n$ or $x \ge n$.

▹ *hint B*

Exercise 4.4 (distance-2 coloring). Let $G = (V, E)$ be a graph. A *distance-2 coloring with k colors* is a function $f : V \to \{1, 2, \ldots, k\}$ with the following property:

$$\mathrm{dist}_G(u, v) \le 2 \text{ implies } f(u) \ne f(v) \text{ for all nodes } u \ne v.$$

Let Δ be a known constant, and let $\mathscr{F}$ be the family of graphs of maximum degree at most Δ. Design a fast distributed algorithm that finds a distance-2 coloring with $O(\Delta^2)$ colors for any graph $G \in \mathscr{F}$ in the LOCAL model.

You can assume that all nodes get the value of n (the number of nodes) as input; also the parameter c in the identifier space is a known constant, so all nodes know the range of unique identifiers.

▹ *hint C*

⋆ **Exercise 4.5** (numeral systems). The fast color reduction algorithm from Section 1.4 is based on the idea of identifying a digit that differs in the *binary* encodings of the colors. Generalize the idea: design an analogous algorithm that finds a digit that differs in the base-k encodings of the colors, for an arbitrary k, and analyze the running time of the algorithm (cf. Exercise 1.6). Is the special case of $k = 2$ the best possible choice?

⋆ **Exercise 4.6** (from bits to sets). The fast color reduction algorithm from Section 1.4 can reduce the number of colors from 2^x to $2x$ in one round in any directed pseudoforest, for any positive integer x. For example, we can reduce the number of colors as follows:

$$2^{128} \to 256 \to 16 \to 8 \to 6.$$

One of the problems is that an iterated application of the algorithm slows down and eventually "gets stuck" at $x = 3$, i.e., at six colors.

In this exercise we will design a faster color reduction algorithm that reduces the number of colors from

$$h(x) = \binom{2x}{x}$$

to $2x$ in one round, for any positive integer x. For example, we can reduce the number of colors as follows:

$$184756 \to 20 \to 6 \to 4.$$

Here

$$\begin{aligned} 184756 &= h(10), \\ 2 \cdot 10 = 20 &= h(3), \\ 2 \cdot 3 = 6 &= h(2). \end{aligned}$$

In particular, the algorithm does not get stuck at six colors; we can use the same algorithm to reduce the number of colors to four. Moreover, at least in this case the algorithm seems to be much more efficient—it can reduce the number of colors from 184756 to 6 in two rounds, while the prior algorithm requires three rounds to achieve the same reduction.

The basic structure of the new algorithm follows the fast color reduction algorithm—in particular, we use one communication round to compute the values $s(v)$ for all nodes $v \in V$. However, the technique for choosing the new color is different: as the name suggests, we will not interpret colors as bit strings but as *sets*.

To this end, let $H(x)$ consist of all subsets

$$X \subseteq \{1,2,\dots,2x\}$$

with $|X| = x$. There are precisely $h(x)$ such subsets, and hence we can find a bijection

$$L\colon \{1,2,\dots,h(x)\} \to H(x).$$

We have $f(v) \neq s(v)$. Hence $L(f(v)) \neq L(s(v))$. As both $L(f(v))$ and $L(s(v))$ are subsets of size x, it follows that

$$L(f(v)) \setminus L(s(v)) \neq \varnothing.$$

We choose the new color $g(v)$ of a node $v \in V$ as follows:

$$g(v) = \min\bigl(L(f(v)) \setminus L(s(v))\bigr).$$

Prove that this algorithm works correctly. In particular, show that $g\colon V \to \{1,2,\dots,2x\}$ is a proper graph coloring of the directed pseudoforest G.

Analyze the running time of the new algorithm and compare it with the old algorithm. Is the new algorithm always faster? Can you prove a general result analogous to the claim of Exercise 1.6?

⋆ **Exercise 4.7** (dominating set approximation). Let Δ be a known constant, and let $\mathscr{F}$ be the family of graphs of maximum degree at most Δ. Design an algorithm that finds an $O(\log \Delta)$-approximation of a minimum dominating set on $\mathscr{F}$ in the LOCAL model.

▹ *hint D*

4.12 Bibliographic Notes

The model of computing is from Linial's [6] seminal paper, and the name LOCAL is from Peleg's [7] book. The additive-group coloring algorithm is due to Barenboim et al. [3]. The effective color reduction algorithm is from Linial [6], and the construction of cover-free families from Barenboim and Elkin [2]. The algorithm of Exercise 4.7 is from Friedman and Kogan [5]. The Bertrand–Chebyshev theorem was first proven by Chebyshev [4]. The proof of Lemma 4.4 follows the proofs of Abraham [1].

4.13 Bibliography

[1] Ittai Abraham. Decentralized thoughts: The marvels of polynomials over a field, 2020. URL: https://decentralizedthoughts.github.io/2020-07-17-the-marvels-of-polynomials-over-a-field/.

[2] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

[3] Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta + 1)$-coloring below Szegedy-Vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proc. 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*, 2018. arXiv:1712.00285, doi:10.1145/3212734.3212769.

[4] Pafnuty Chebyshev. Mémoire sur les nombres premiers. *Journal de mathématiques pures et appliquées*, 17(1):366–390, 1852.

[5] Roy Friedman and Alex Kogan. Deterministic dominating set construction in networks with bounded degree. In *Proc. 12th International Conference on Distributed Computing and Networking (ICDCN 2011)*, 2011. doi:10.1007/978-3-642-17679-1_6.

[6] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[7] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[8] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

4.14 Hints

A. Solve small problem instances by brute force and focus on the case of long cycles. In a long cycle, use a graph coloring algorithm to find a 3-coloring, and then use the 3-coloring to construct a maximal independent set. Observe that a maximal independent set partitions the cycle into short fragments (with 2–3 nodes in each fragment).

   Apply the same approach recursively: interpret each fragment as a "supernode" and partition the cycle that is formed by the supernodes into short fragments, etc. Eventually, you have partitioned the original cycle into *long* fragments, with dozens of nodes in each fragment.

   Find an optimal vertex cover within each fragment. Make sure that the solution is feasible near the boundaries, and prove that you are able to achieve the required approximation ratio.

B. Adapt the basic idea of the greedy color reduction algorithm—find local maxima and choose appropriate colors for them—but pay attention to the stopping conditions and low-degree nodes. One possible strategy is this: a node becomes active if its current color is a local maximum among those neighbors that have not yet stopped; once a node becomes active, it selects an appropriate color and stops.

C. Given a graph $G \in \mathscr{F}$, construct a virtual graph $G^2 = (V, E')$ as follows: $\{u, v\} \in E'$ if $u \neq v$ and $\mathrm{dist}_G(u, v) \leq 2$. Prove that the maximum degree of G^2 is $O(\Delta^2)$. Simulate a fast graph coloring algorithm on G^2.

D. First, design (or look up) a greedy *centralized* algorithm achieves an approximation ratio of $O(\log \Delta)$ on $\mathscr{F}$. The following idea will work: repeatedly pick a node that *dominates* as many new nodes as possible—here a node $v \in V$ is said to dominate all nodes in $\text{ball}_G(v, 1)$. For more details, see a textbook on approximation algorithms, e.g., Vazirani [8].

Second, show that you can *simulate* the centralized greedy algorithm in a distributed setting. Use the algorithm of Exercise 4.4 to construct a distance-2 coloring. Prove that the following strategy is a faithful simulation of the centralized greedy algorithm:

- For each possible value $i = \Delta + 1, \Delta, \dots, 2, 1$:
    - For each color $j = 1, 2, \dots, O(\Delta^2)$:
        - Pick all nodes $v \in V$ that are of color j and that dominate i new nodes.

The key observation is that if $u, v \in V$ are two distinct nodes of the same color, then the set of nodes dominated by u and the set of nodes dominated by v are disjoint. Hence it does not matter whether the greedy algorithm picks u before v or v before u, provided that both of them are equally good from the perspective of the number of new nodes that they dominate. Indeed, we can equally well pick both u and v simultaneously in parallel.

4.15 Appendix: Finite Fields

For our purposes, finite field of size q can be seen as the set $\{0, \dots, q-1\}$ equipped with modular arithmetic, for any prime q. Fields support addition, subtraction, multiplication, and division with the usual rules. We denote the finite field with q elements (also known as a Galois field) by $\text{GF}(q)$.

Our proofs will use the following two properties of finite fields.

(a) Each element a of the field has a unique multiplicative inverse element, denoted by a^{-1}, such that $a \cdot a^{-1} = 1$.

(b) The product ab of two elements is zero if and only if $a = 0$ or $b = 0$.

We can define polynomials over GF(q). A polynomial $f[X]$ of degree d can be represented as

$$f_0 + f_1X + f_2X^2 + \cdots + f_dX^d,$$

where the coefficients f_i are elements of GF(q). A polynomial is non-trivial if there exists some $f_i \neq 0$. An element $a \in \mathrm{GF}(q)$ is a zero of a polynomial f if $f(a) = 0$.

Proof of Lemma 4.4. We will prove the lemma by proving a related statement: any non-trivial polynomial of degree d has at most d zeros. Since $f(x) - g(x)$ is a polynomial of degree at most d, Lemma 4.4 follows.

The proof is by induction on d. Let $f[X] = f_0 + f_1X$ denote an arbitrary polynomial of degree 1 over some finite field of size q. Since each element a of a field has a unique inverse a^{-1}, there is a unique zero of $f[X]$: $X = -(f_0)(f_1)^{-1}$.

Now assume that $d \geq 2$ and that the claim holds for smaller degrees. If polynomial f has no zeros, the claim holds. Therefore assume that f has at least one zero $a \in \mathrm{GF}(q)$. We will show that there exists a polynomial g of degree $d - 1$ such that $f = (X - a)g$. By the induction hypothesis g has at most $d - 1$ zeros, $X - a$ has one zero, and we know that the product equals zero if and only if either $X - a = 0$ or $g[X] = 0$.

We show that g exists by induction. If $d = 1$, we can select $a = -(f_0)(f_1)^{-1}$ and $g = f_1$ to get $f[X] = (X + (f_0)(f_1)^{-1})f_1$.

For $d \geq 2$, we again make the induction assumption. Define

$$f' = f - f_dX^{d-1}(X - a),$$

where f_d is the dth coefficient of f. This polynomial has degree less than d, since the terms of degree d cancel out. We also have that $f'(a) = 0$ since $f(a) = 0$ by assumption. By induction hypothesis there exists a g' such that $f' = (X - a)g'$ and degree of g' is at most $d - 2$. By substituting $f' = (X - a)g'$ we get

$$f = (X - a)g' + (X - a)f_dX^{d-1} = (X - a)(g' + f_dX^{d-1}).$$

Therefore $f = (X - a)g$ for the polynomial $g = g' + f_d X^{d-1}$, a polynomial of degree at most $d - 1$. □