Exact bounds for distributed graph colouring

Joel Rybicki

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, Aalto University

Department of Algorithms and Complexity,
Max Planck Institute for Informatics

joel.rybicki@aalto.fi

Jukka Suomela

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, Aalto University

jukka.suomela@aalto.fi

Abstract. We prove exact bounds on the time complexity of distributed graph colouring. If we are given a directed path that is properly coloured with n colours, by prior work it is known that we can find a proper 3-colouring in $\frac{1}{2}\log^*(n) \pm O(1)$ communication rounds. We close the gap between upper and lower bounds: we show that for infinitely many n the time complexity is precisely $\frac{1}{2}\log^* n$ communication rounds.

1 Introduction

One of the key primitives in the area of distributed graph algorithms is *graph colouring in directed paths*. This is a fundamental symmetry-breaking task, widely studied since the 1980s—it is used as a subroutine in numerous efficient distributed algorithms, and it also serves as a convenient starting point in many lower-bound proofs. In the 1990s it was already established that the distributed computational complexity of this problem is $\frac{1}{2}\log^*(n) \pm O(1)$ communication rounds [3,13]. We are now able to give *exact* bounds on the distributed time complexity of this problem, and the answer turns out to take a surprisingly elegant form:

Theorem 1. *For infinitely many values of n, it takes exactly $\frac{1}{2}\log^* n$ rounds to compute a 3-colouring of a directed path.*

1.1 Problem Setting

Throughout this work we focus on *deterministic* distributed algorithms. As is common in this context, what actually matters is not the number of nodes but the range of their labels. For the sake of concreteness, we study precisely the following problem setting:

> We have a path or a cycle with any number of nodes, and the nodes are properly coloured with colours from $[n] = \{1, 2, \ldots, n\}$.

The techniques that we present in this work can also be used to analyse other variants of the problem—for example, a cycle with n nodes that are labelled with some permutation of $[n]$, or a path with at most n nodes that are labelled with unique identifiers from $[n]$. However, the exact bounds on the time complexity will slightly depend on such details.

We will assume that there is a globally consistent orientation in the path: each node has at most one predecessor and at most one successor. Our task is to find a proper colouring of the path with c colours, for some number $c \geq 3$. We will call this task *colour reduction from n to c*.

We will use the following model of distributed computing. Each node of the graph is a computational entity. Initially, each node knows the global parameters n and c, its own label from $[n]$, its degree, and the orientations of its incident edges. Computation takes place in synchronous communication rounds. In each round, each node can send a message to each of its neighbours, receive a message from each of its neighbours, update its state, and possibly stop and output its colour. The *running time* of an algorithm is defined to be the number of communication rounds until all nodes have stopped. We will use the following notation:

- $C(n, c)$ is the time complexity of colour reduction from n to c.
- $T(n, c)$ is the time complexity of colour reduction from n to c if we restrict the algorithm so that a node can only send messages to its successor. We call such an algorithm *one-sided*, while unrestricted algorithms are *two-sided*.

We can compose colour reduction algorithms, yielding $C(a, c) \leq C(a, b) + C(b, c)$ and $T(a, c) \leq T(a, b) + T(b, c)$ for any $a \geq b \geq c$. It is easy to see (shown in Lemma 2) that

$$C(n, c) = \lceil T(n, c)/2 \rceil.$$

We will be interested primarily in $C(n, c)$, but function $T(n, c)$ is much more convenient to analyse when we prove upper and lower bounds.

1.2 Prior Work

The asymptotically optimal bounds of

$$\log^*(n) - O(1) \leq T(n,3) \leq \log^*(n) + O(1)$$

are covered in numerous textbooks and courses on distributed and parallel computing [2,4,17,19,20]. The proof is almost unanimously based on the following classical results:

Cole–Vishkin colour reduction (CV): The upper bound was presented in the modern form by Goldberg, Plotkin, and Shannon [9] and it is based on the technique first introduced by Cole and Vishkin [3]. The key ingredients are a fast colour reduction algorithm that shows that $T(2^k, 2k) \leq 1$ for any $k \geq 3$, and a slow colour reduction algorithm that show that $T(k+1, k) \leq 2$ for any $k \geq 3$. By iterating the fast colour reduction algorithm, we can reduce the number of colours from n to 6 in $\log^*(n) \pm O(1)$ rounds, and by iterating the slow colour reduction algorithm, we can reduce the number of colours from 6 to 3 in 6 rounds (with one-sided algorithms).

Linial's lower bound: The lower bound is the seminal result by Linial [13]. The key ingredient is a speed-up lemma that shows that $T(n, 2^c) \leq T(n, c) - 1$ when $T(n, c) \geq 1$. By iterating the speed-up lemma for $\log^*(n) - 3$ times, we have $T(n, 4) \geq T(n, k) + \log^*(n) - 3$ for a $k < n$. Clearly $T(n, 3) \geq T(n, 4)$ and $T(n, k) \geq 1$, and hence $T(n, 3) \geq \log^*(n) - 2$.

In the upper bound, many sources—including the original papers by Cole and Vishkin and Goldberg et al.—are happy with the asymptotic bounds of $\log^*(n) + O(1)$ or $O(\log^* n)$. However, there are some sources that provide a more careful analysis. The analysis by Barenboim and Elkin [2] yields $T(n, 3) \leq \log^*(n) + 9$, and the analysis in the textbook by Cormen et al. [4] yields $T(n, 3) \leq \log^*(n) + 7$. In our lecture course [19] we had an exercise that shows how to push it down to

$$T(n, 3) \leq \log^*(n) + 6.$$

In the lower bound, there is less variation. Linial's original proof [13] yields $T(n, 3) \geq \log^*(n) - 3$, and many sources [2, 11, 19] prove a bound of

$$T(n, 3) \geq \log^*(n) - 2.$$

On the side of lower bounds, nothing stronger than Linial's result is known. There are alternative proofs based on Ramsey's theorem [5] that yield the same asymptotic bound of $T(n, 3) = \Omega(\log^* n)$, but the constants one gets this way are worse than in Linial's proof.

On the side of upper bounds, however, there is an algorithm that is strictly better than CV: **Naor–Stockmeyer colour reduction (NS)** [15]. While CV yields $T(2^k, 2k) \leq 1$ for any $k \geq 3$, NS yields a strictly stronger claim of $T(\binom{2k}{k}, 2k) \leq 1$ for any $k \geq 2$. However, the exact bounds that we get from NS are apparently not analysed anywhere, and their algorithm is hardly ever mentioned in the literature. Hence the state of the art appears to be

$$\log^*(n) - 2 \leq T(n, 3) \leq \log^*(n) + 6,$$
$$\frac{1}{2}\log^*(n) - 1 \leq C(n, 3) \leq \frac{1}{2}\log^*(n) + 3.$$

Note that we have $\log^* n \leq 5$ for all $n < 10^{19728}$, and hence in practice the constant term 6 dominates the term $\log^* n$ in the upper bound.

1.3 Contributions

In this work we derive *exact bounds* on $C(n, 3)$ for infinitely many values of n, and near-tight bounds for all values of n. We prove that for infinitely many values of n

$$C(n, 3) = \frac{1}{2} \log^* n,$$

and for all sufficiently large values of n

$$\log^*(n) - 1 \leq T(n, 3) \leq \log^*(n) + 1.$$

With $C(n, 3) = \lceil T(n, 3)/2 \rceil$ this gives a near-complete picture of the exact complexity of colouring directed paths. The key new techniques are as follows:

1. We give a new analysis of NS colour reduction.
2. We give a new lower-bound proof that is strictly stronger than Linial's lower bound.
3. We show that *computational techniques* can be used to prove not only upper bounds but also lower bounds on $T(n, c)$, also for the case of a general n and not just for fixed small values of n and c. We introduce *successor graphs* $\mathcal{S}_i$ that are defined so that a graph colouring of $\mathcal{S}_i$ with a small number of colours implies an improved bound on $T(n, 3)$.

This work focuses on colour reduction, i.e., the setting in which we are given a proper colouring as an input. Our upper bounds naturally apply directly in more restricted problems (e.g., the input labels are unique identifiers). Our lower bounds results do not hold directly, but the key techniques are still applicable: in particular, the successor graph technique can be used also in the case of unique identifiers.

1.4 Applications

Graph colouring in paths, and the related problems of graph colouring in rooted trees and directed pseudoforests, are key symmetry-breaking primitives that appear as subroutines in numerous distributed algorithms for various graph problems [1, 5, 8, 9, 12, 16].

One of the most direct application of our results is related to colouring *trees*: In essence, colour reduction from n to c in trees with *arbitrary* algorithms is the same problem as colour reduction from n to c in paths with *one-sided* algorithms. Informally, in the worst case the children contain all possible coloured subtrees and hence "looking down" in the tree is unhelpful, and we can equally well restrict ourselves to "looking up" towards the root. Hence our bounds on $T(n, 3)$ can be directly interpreted as bounds on colour reduction from n to 3 in trees.

The bounds have also applications outside distributed computing. A result by Fich and Ramachandran [6] demonstrates that bounds on $C(n, 3)$ have direct implications in the context of *decision trees* and *parallel computing*.

Indeed, the fastest known *parallel* algorithms for colouring linked lists are just adaptations of CV and NS colour reduction algorithms. These algorithms reduce the number of colours very rapidly to a relatively small number (e.g., dozens of colours), and the key bottleneck has been pushing the number of colours down to 3. In particular, reducing the number of colours down to 3 with state-of-the-art algorithms has been much

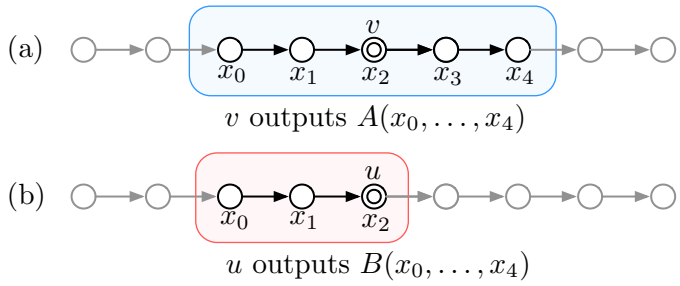

Figure 1: The difference of two-sided and one-sided algorithms. (a) A two-sided algorithm A that runs for 2 rounds. (b) A one-sided algorithm B that runs for 2 rounds.

more expensive than reducing it to 4, but this phenomenon has not been understood so far. Prior bounds on $T(n, c)$ have not been able to show that the case of $c = 3$ is necessarily more expensive than $c = 4$. Our improved bounds are strong enough to separate $T(n, 4)$ and $T(n, 3)$.

From the perspective of practical algorithm engineering and programming, this work shows that we should avoid CV colour reduction, but we can be content with NS colour reduction; the former incurs a significant overhead (e.g., in terms of linear scans over the data in parallel computing), but the latter is near-optimal.

2 Preliminaries

Sets and Functions. For any positive integer k, we use $[k]$ to denote the set $\{1, 2, \ldots, k\}$. For any set X, we use $2^X = \{Y \subseteq X\}$ to denote the powerset of X. Define the *iterated logarithm* as

$$\log^{(0)}(x) = x,$$
$$\log^{(i+1)}(x) = \log^{(i)}(\log x) \text{ for all } i \geq 0.$$

In this work, all logarithms are in base 2. Moreover, the *log-star* function is

$$\log^* x = \min\{i : \log^{(i)} x \leq 1\}.$$

Finally, we define the *tetration*, or a power tower, with base 2 as

$$^0 2 = 1,$$
$$^{i+1}2 = 2^{(^i 2)} \text{ for all } i \geq 0.$$

Algorithms. In this work, we focus on algorithms that run on directed paths. We distinguish between two-sided and one-sided algorithms; see Figure 1. *Two-sided algorithms* correspond to the usual notion of an algorithm in the LOCAL model: an algorithm running for t rounds has to decide on its output using the information available at most t hops away. Formally, a two-sided c-colouring algorithm corresponds to a function

$$A \colon [n]^{2t+1} \to [c].$$

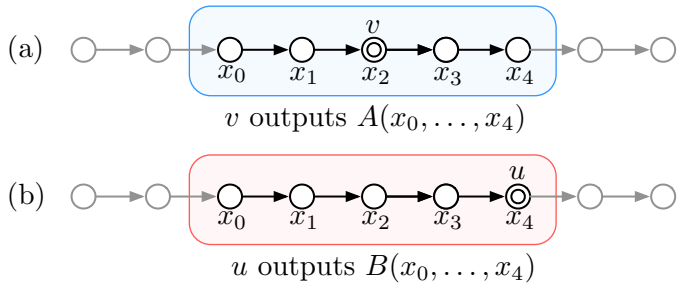

Figure 2: The correspondence between two-sided and one-sided algorithms. (a) A two-sided algorithm A that runs for 2 rounds. (b) A one-sided algorithm that runs in 4 rounds. Both nodes see the same information, so v can easily simulate B and u can simulate A.

Moreover, as A outputs a proper colouring, the function satisfies $A(x_0, \ldots, x_{2t}) \neq A(x_1, \ldots, x_{2t+1})$ when $x_i \neq x_{i+1}$ for all $i \geq 0$.

In contrast to two-sided algorithms, *one-sided algorithms* are algorithms in which nodes can only send messages to successors. Therefore, a one-sided algorithm that runs in t rounds can only gather information from at most t *predecessors*. Formally, a one-sided c-colouring algorithm B that runs for t steps corresponds to a function

$$B\colon [n]^{t+1} \to [c],$$

which satisfies $B(x_0, \ldots, x_t) \neq B(x_1, \ldots, x_{t+1})$ when $x_i \neq x_{i+1}$ for all $i \geq 0$.

It is now easy to see that $C(n, c) = \lceil T(n, c)/2 \rceil$ holds. Intuitively, the connection is straightforward. For example, Figure 2 illustrates how a t-time two-sided algorithm can gather the same information as a $2t$-time one-sided algorithm. For the sake of completeness, we now prove this formally.

Lemma 2. $C(n, c) = \lceil T(n, c)/2 \rceil$.

Proof. First, we show that $T(n, c) \leq 2C(n, c)$. Let $t = C(n, c)$ and $A\colon [n]^{2t+1} \to [c]$ be a two-sided c-colouring algorithm that runs in time t. We construct a one-sided c-colouring algorithm that runs in time $2t$. Recall that a one-sided algorithm can only receive messages from predecessors. Initially, every node sends its own colour to its successor. Then for $2t - 1$ rounds we send the colour received from the predecessor to the successor—in the case that a node has no predecessors, the node can simply simulate a properly coloured path preceding it. After $2t$ rounds the node knows its own colour and the colours of its $2t$ predecessors, that is, a vector $(x_0, \ldots, x_{2t}) \in [n]^{2t+1}$. Outputting the value $A(x_0, \ldots, x_{2t})$ yields a proper colouring.

Second, we show that $C(n, c) \leq \lceil T(n, c)/2 \rceil$. Let $t = \lceil T(n, c)/2 \rceil$ and $B\colon [n]^{T(n,c)+1} \to [c]$ a one-sided algorithm that only receives messages from predecessors. Every node sends its colour to both neighbours and then forwards any messages in the $t - 1$ subsequent rounds. As $2t \geq T(n, c)$, after t rounds every node knows the colours $(x_0, \ldots, x_{T(n,c)})$ in its local neighbourhood. Now the node can output $B(x_0, \ldots, x_{T(n,c)})$ which gives a proper colouring.

Finally, since the time complexity has to be integral—there are no "half-rounds"—we get that $C(n, c) = \lceil T(n, c)/2 \rceil$. □

3 The Upper Bound

In this section, we bound $T(n, c)$ from above. To do this, we analyse the Naor–Stockmeyer (NS) colour reduction algorithm [15]. The NS algorithm is one-sided, thus yielding upper bounds for $T(n, c)$.

Let us first recall the NS colour reduction algorithm. Let $n \leq \binom{2k}{k}$ for some $k \geq 2$ and fix an injection $f\colon [n] \to X$, where $X = \{Y \subseteq [2k] : |Y| = k\}$. That is, we interpret all colours from $[n]$ as distinct k-subsets of $[2k]$.

The algorithm works as follows. First, all nodes send their colour to the successor. Then a node with colour v receiving colour u from its predecessor will output

$$A(u, v) = \min f(u) \setminus f(v).$$

It is easy to show that if $u \neq v \neq w$, then $A(u, v) \in [2k]$ and $A(u, v) \neq A(v, w)$ holds. Thus, A is a one-sided colour reduction algorithm that reduces the number of colours from $\binom{2k}{k}$ to $2k$ colours in one round and we have that $T\big(\binom{2k}{k}, 2k\big) = 1$ for any $k \geq 2$.

The above algorithm cannot reduce the number of colours below 4. To reduce the number of colours from four to three, we can use the following one-sided algorithm B that outputs

$$B(u, v, w) = \begin{cases} \min\{1, 2, 3\} \setminus \{u, w\} & \text{if } v = 4, \\ v & \text{otherwise.} \end{cases}$$

The algorithm uses two rounds and this is optimal by Lemma 8 in Section 4.

We now show the following upper bounds for $T(n, c)$ using the NS colour reduction algorithm.

Lemma 3. *The function T satisfies the following:*

(a) $T\big(\frac{3}{2} \cdot 2^c,\ \frac{3}{2} \cdot c\big) = 1$ *for any $c = 4h$, where $h > 1$,*

(b) $T\big(\frac{3}{2} \cdot {}^{r+4}2,\ \frac{3}{2} \cdot {}^{4}2\big) \leq r$ *for any $r \geq 0$,*

(c) $T\big(\frac{3}{2} \cdot {}^{4}2,\ 3\big) \leq 5$.

Proof.

(a) As discussed, the NS colour reduction algorithm shows that $T\big(\binom{2k}{k}, 2k\big) = 1$ for $k \geq 2$. Recall the following bound for the central binomial coefficent

$$\binom{2k}{k} \geq \frac{4^k}{\sqrt{4k}}$$

and let $2k = 3c/2$. Since $c \geq 8$ it follows that

$$\binom{2k}{k} \geq \frac{(2 \cdot 2)^{3c/4}}{\sqrt{3c}} = \frac{2^{c/2}}{\sqrt{3c}} \cdot 2^c > \frac{3}{2} \cdot 2^c.$$

(b) To show the claim, it suffices to apply part (a) for r times.

(c) As $\binom{20}{10} > \frac{3}{2} \cdot {}^{4}2$, we can reduce the number of colours to 4 in three rounds as follows: $\binom{20}{10} \rightsquigarrow \binom{6}{3} \rightsquigarrow \binom{4}{2} \rightsquigarrow 4$. By Lemma 8, the remaining two rounds can be used to remove the fourth colour. □

Theorem 4. $T({}^{h}2, 3) \leq T({}^{h}2 + 1, 3) \leq h + 1$ *holds for any $h > 1$.*

Proof. The cases $2 \leq h \leq 4$ follow from the proof of Lemma 3c. Suppose $h = r + 4$ for some $r > 0$. By Lemma 3b and c we can get a 3-colouring in $r + 5 = h + 1$ rounds. □

4 The Lower Bound

In this section, we give a new lower bound for the time complexity of one-sided colour reduction algorithms. The proof follows the basic idea of Linial's proof [13] adapted to the case of colour reduction, but we show a new lemma that can be used to tighten the bound.

The proof is structured as follows. First, we show that $T(n, 2^c - 2) \leq T(n, c) - 1$, that is, given a c-colouring algorithm, we can devise a faster algorithm that uses at most $2^c - 2$ colours; this is just a minor tightening of the usual standard bound, and should be fairly well-known. Second, we prove that a fast 3-colouring algorithm implies a fast 16-colouring algorithm, more precisely, $T(n, 16) \leq T(n, 3) - 2$; this is the key contribution of this section. Together these yield the following new bound:

Theorem 5. *For any $h > 1$, we have $T({}^h2, 3) \geq h$.*

4.1 The Speed-up Lemma

Lemma 6. *If $T(n, c) \geq 1$, then $T(n, 2^c - 2) \leq T(n, c) - 1$.*

Proof. Let $t = T(n, c)$ and $A\colon [n]^{t+1} \to [c]$ be a one-sided c-colouring algorithm. We will construct a faster one-sided algorithm B as follows. Consider a node u and its successor v. In $t - 1$ rounds, node u can find out the colours of its $t - 1$ predecessors and its own colour, that is, some vector $(x_0, \ldots, x_{t-1}) \in [n]^t$. In particular, node u now knows what information node v can gather in t rounds *except* the colour of v since A is one-sided. However, u can enumerate all the possible outputs of v which give the set

$$B(x_0, \ldots, x_{t-1}) = \big\{A(x_0, \ldots, x_{t-1}, y)\colon y \neq x_{t-1}, y \in [n]\big\} \subseteq [c].$$

Clearly $B(x_0, \ldots, x_{t-1}) \neq \emptyset$. We also have $B(x_0, \ldots, x_{t-1}) \neq [c]$: For the sake of contradiction, suppose otherwise. This would imply that v could output any value in $[c]$. In particular, if u outputs $A(z, x_0, \ldots, x_{t-1}) = a$ for some $z \in [n]$, we could pick $y \in [n]$ such that $A(x_0, \ldots, x_{t-1}, y) = a$ as well. However, this would contradict the fact that A was a colouring algorithm. Hence there exists an injection f that maps any possible set $B(\cdot)$ to a value in $[2^c - 2]$.

It remains to argue that no two adjacent nodes construct the same set. Suppose a node u outputs set X and its successor v also outputs X. Now we can pick $k \in X$ such that

$$A(x_0, \ldots, x_{t-1}, y) = k = A(x_1, \ldots, x_{t-1}, y, y')$$

for some $x_{t-1} \neq y \neq y'$ contradicting that A outputs a proper colouring. Therefore, $f \circ B$ is a one-sided $(2^c - 2)$-colouring algorithm that runs in time $t - 1 = T(n, c) - 1$. □

Lemma 7. *For any $r > 0$, we have $T({}^{r+3}2, 16) \geq r + 1$.*

Proof. Fix $r > 0$. We repeatedly apply Lemma 6. Now suppose we have an algorithm that reduces the number of colours from n to $16 = {}^3 2$ in r rounds. That is, $T(n, {}^3 2) \leq r$ holds for some $n \geq 3$. From Lemma 6 it follows that

$$\begin{aligned} T(n, {}^3 2) \leq r &\implies T(n, {}^4 2 - 2) \leq r - 1 \\ &\implies \cdots \\ &\implies T(n, {}^{3+r} 2 - 2) \leq 0, \end{aligned}$$

but as $T(k, k-1) \geq 1$ for any k it follows that $n < {}^{3+r}2$. Thus, $T({}^{r+3}2, 16) \geq r + 1$. □

4.2 Proof of Theorem 5

In addition to the speed-up lemma, we need a few more lemmas that bound $T(n, 3)$ below for small values of n.

Lemma 8. *$T(4, 3) \geq 2$.*

Proof. Let $B' \colon (u, v) \to \{1, 2, 3\}$ be a one-sided 3-colouring algorithm that runs in one round. Now B' yields a partitioning of the possible input pairs (u, v) where $u \neq v$. It is simple to check that there always exists a pair (u, v) with $u \neq v$ such that there also exists some $w \neq v$ satisfying $B'(u, v) = B'(v, w)$. □

Lemma 9. *$T(16, 3) \geq 3$.*

Proof. As observed by Linial [13], we can show $C(n, c) = t$ if the so-called *neighbourhood graph* $\mathcal{N}_{n,t}$ has a chromatic number of c. While Linial analytically bounded the chromatic number of such graphs, we can also compute their chromatic numbers exactly for small values of n, c, and t; see [18] for a detailed discussion. We use the latter technique to show the claimed bound. That is, the neighbourhood graph $\mathcal{N}_{7,1}$ is not 3-colourable.

The neighbourhood graph $\mathcal{N}_{7,1} = (V, E)$ is defined as follows. The set of vertices is

$$V = \{(x_0, x_1, x_2) \in [n]^3 : x_0 \neq x_1 \neq x_2, x_0 \neq x_2\},$$

where $n = 7$ and the set of edges is

$$E = \{\{u, v\} : u, v \in V, u = (x_0, x_1, x_2), v = (x_1, x_2, x_3)\}.$$

It is easy to check with a computer (e.g. using any off-the-shelf SAT or an IP solver) that the graph $\mathcal{N}_{7,1}$ is not 3-colourable. Therefore, $C(7, 3) > 1$ and in particular $T(16, 3) \geq T(7, 3) > 2$. □

To get a lower bound for 3-colouring, we show in the following sections that the existence of a t-time one-sided 3-colouring algorithm implies a $(t - 2)$-time one-sided 16-colouring algorithm.

Lemma 10. *For any $n \geq 16$, it holds that $T(n, 16) \leq T(n, 3) - 2$.*

Now we have all the results for showing the lower bound.

Theorem 5. *For any $h > 1$, we have $T(^h 2, 3) \geq h$.*

Proof. The cases $r = 2$ and $r = 3$ follow from Lemmas 8 and 9. For the remaining cases, let $h = r + 3$ for some $r > 0$. Suppose $T(^h 2, 3) = T(^{r+3} 2, 3) < h$. Then by Lemma 10 we would get that $T(^{r+3} 2, 16) < h - 2 = r + 1$ which contradicts Lemma 7. □

4.3 Proof of Lemma 10 via Successor Graphs

To prove Lemma 10, we analyse the chromatic number of so-called *successor graphs*—a notion similar to Linial's neighbourhood graphs [13]. In the following, given a binary relation R, we will write $x \in R(y)$ to mean $(y, x) \in R$.

Colouring Relations. Suppose $A = A_0$ is a one-sided 3-colouring algorithm that runs in t rounds. Let $A_1, \ldots, A_t$ denote the one-sided algorithms given by iterating Lemma 6 and $C_{k+1} \subseteq 2^{C_k}$ be the set of colours output by algorithm A_{k+1}.

In the following, let $t' = t - k$. Define the *potential successor relation* $S_k \subseteq C_k \times C_k$ to be a binary relation such that $(x, y) \in S_k$ if there exists $x_0, \ldots, x_{t'}$ where $x_i \neq x_{i+1}$ such that

$$A_k(x_0, \ldots, x_{t'-1}) = x \text{ and } A_k(x_1, \ldots, x_{t'}) = y.$$

That is, in the output of algorithm A_k there can be an x-coloured node with a successor of colour y. Moreover, define the *output relation* $R_k \subseteq C_k \times C_{k+1}$ such that $(x, X) \in R_k$ if

$$A_k(x_0, \ldots, x_{t'-2}, x) = X$$

for some $x_0, \ldots, x_{t'-2}$ where $x_i \neq x_{i+1}$. That is, a node with colour x can output colour X when executing A_{k+1}. From the construction of A_{k+1} given in Lemma 6, we get that $R_k = \{(x, X) : X \subseteq S_k(x), X \neq \emptyset\}$.

Lemma 11. *Suppose $X \in R_k(x)$, $Y \in R_k(y)$, and $y \in X$ for some $x, y \in C_k$, then $(X, Y) \in S_{k+1}$ holds. Moreover, the converse holds.*

Proof. As we have $y \in X \subseteq S_k(x)$, this means that a node with colour x may have a successor of colour y after executing algorithm A_k. Moreover, as $X \in R_k(x)$ and $Y \in R_k(y)$ hold, then a node with colour x may output X and node with colour y may output Y when executing A_{k+1}. Thus, after executing A_{k+1} we may have a node with colour X that has a successor with colour Y. Therefore, $(X, Y) \in S_{k+1}$.

To show the converse, suppose that $(X, Y) \in S_{k+1}$, that is, in some output of A_{k+1} a node u with colour X having a successor v with colour Y. Now there must exist some colour x that $X \in R_k(x)$ and some colour y such that $Y \in R_k(y)$. As v is a successor of u, the algorithm A_{k+1} outputs a set X consisting of all possible colours for any successor of u, and thus, we have $y \in X$. □

Successor Graphs. For any choice of $A = A_0$, we can construct the successor relation S_k and using this relation, we can define the *successor graph* of A to be the graph $\mathcal{S}_k(A) = (C_k, E_k)$, where $E_k = \{\{x, y\} : (x, y) \in S_k\}$. These graphs have the following property:

Lemma 12. *Let $\mathcal{S}_k = (C_k, S_k)$ be the successor graph of A, and let t be the running time of A. If $f \colon C_k \to [\chi]$ is a proper colouring of $\mathcal{S}_k$, then $f \circ A_k$ is a one-sided χ-colouring algorithm that runs in $t - k$ rounds. That is, $T(n, \chi) \leq t - k$.*

Proof. Let u be the predecessor of v on a directed path. Now by definition,

$$\begin{aligned} &A_k(x_0, \ldots, x_{t-1}, u) = x \neq y = A_k(x_1, \ldots, x_{t-1}, u, v) \\ &\implies (x, y) \in S_k \implies f(x) \neq f(y). \end{aligned}$$

Therefore, $f \circ A_k$ is a one-sided χ-colouring algorithm. □

In the next section, we show the following lemma from which Lemma 10 follows.

Lemma 13. *For any t-time 3-colouring algorithm A, the successor graph $\mathcal{S}_2(A)$ can be coloured with* 16 *colours.*

In particular, this holds for an optimal algorithm A with a running time of $t = T(n, 3)$. Together with Lemma 12, this implies Lemma 10. We next show how to prove Lemma 13 in two ways: with computers, and without them.

4.4 A Human-Readable Proof of Lemma 13

We start by giving a traditional human-readable proof for Lemma 13. That is, we argue that for any one-sided 3-colouring algorithm $A = A_0$ the successor graph $\mathcal{S}_2(A)$ can be coloured with 16 colours. Later in Section 4.5, we give a computational proof of the same result. In the following, we fix A and denote $\mathcal{S}_2 = \mathcal{S}_2(A)$ for brevity.

Structural Properties. We start with the following observations.

Remark 1. Sets C_0 and C_1 satisfy

$$C_0 \subseteq \{1,2,3\},$$
$$C_1 \subseteq \big\{\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\}\big\}.$$

Remark 2. Relation S_1 satisfies

$$S_1(i) \subseteq \big\{X \in C_1 : i \notin X\big\},$$
$$S_1(\{i,j\}) \subseteq \big\{X \in C_1 : \{i,j\} \not\subseteq X\big\}.$$

Remark 3. Consider any $X \subseteq C_1$ with $\big\{\{1,2\},\{1,3\},\{2,3\}\big\} \subseteq X$. Then there is no $x \in C_1$ with $X \subseteq S_1(x)$. Therefore A_2 cannot output colour X, and hence $X \notin C_2$.

Hence graph $\mathcal{S}_2$ has $|C_2| \leq 55$ nodes: out of the $2^6 = 64$ candidate colours, we can exclude the empty set and 8 other sets identified in Remark 3. We will now partition the remaining nodes in 16 colour classes (independent sets).

Colour Classes. There are four types of colour classes. First, for each $\emptyset \neq X \subseteq [3]$ we define a singleton colour class

$$\mathcal{X}_0(X) = \Big\{\big\{\{x\} : x \in X\big\}\Big\},$$

that is, an independent set of size 1. Then for each triple

$$(i,j,k) \in \big\{(1,2,3),\ (1,3,2),\ (2,3,1)\big\}$$

we have three colour classes:

$$\mathcal{X}_1(i,j,k) = \Big\{X \in C_2 : \big\{\{i,j\},\{i,k\}\big\} \subseteq X \subseteq \big\{\{i,j\},\{i,k\},\{i\},\{j\},\{k\}\big\}\Big\}$$
$$\mathcal{X}_2(i,j,k) = \Big\{X \in C_2 : \big\{\{i,j\},\{k\}\big\} \subseteq X \subseteq \big\{\{i,j\},\{i\},\{j\},\{k\}\big\}\Big\},$$
$$\mathcal{X}_3(i,j,k) = \Big\{X \in C_2 : \big\{\{i,j\}\big\} \subseteq X \subseteq \big\{\{i,j\},\{i\},\{j\}\big\}\Big\}.$$

In total, there are 7 singleton colour classes, and 3×3 other colour classes, giving in total 16 colour classes. Figure 3 shows the complement of a supergraph of $\mathcal{S}_2$; each of the above colour classes correspond to a clique in the complement graph.

It can be verified that each of the 55 possible nodes of $\mathcal{S}_2$ is included in exactly one of the colour classes. It remains to be shown that each colour class is indeed an independent set of $\mathcal{S}_2$.

The singleton classes form independent sets trivially. We handle each type of the remaining colour classes separately. Recall that there is an edge $\{X, Y\}$ in $\mathcal{S}_2$ if either $X \in S_2(Y)$ or $Y \in S_2(X)$.

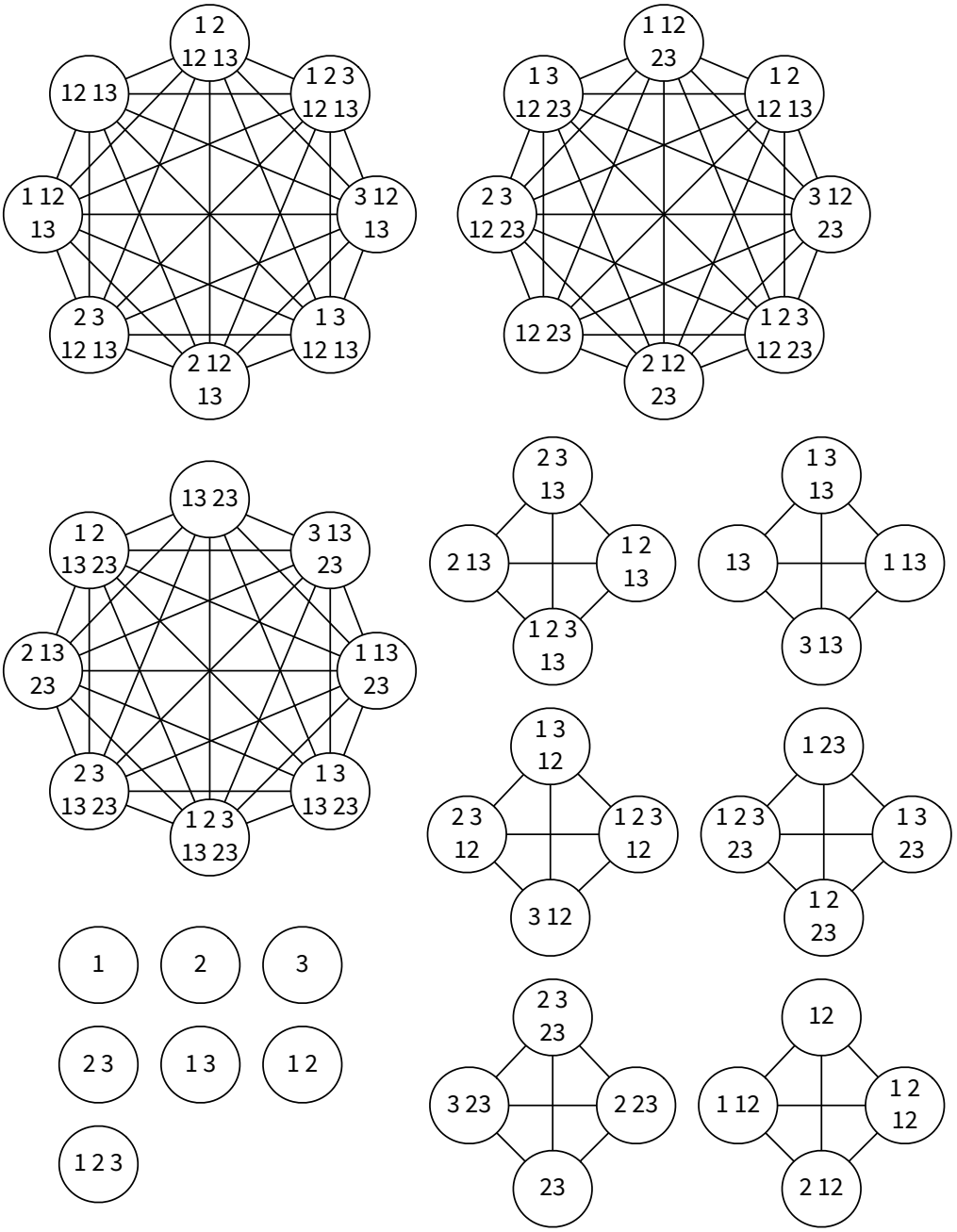

Figure 3: This illustrations shows the *complement* of a graph we call $\mathcal{S}_2^*$. For any algorithm A, the successor graph $\mathcal{S}_2(A)$ is a subgraph of $\mathcal{S}_2^*$, and hence, a proper colouring of $\mathcal{S}_2^*$ is a proper colouring of $\mathcal{S}_2(A)$. Each clique in the figure corresponds to a colour class in $\mathcal{S}_2^*$. We use a shorthand notation: for example, the circle labelled with "1 2 12" is the node $\{\{1\}, \{2\}, \{1, 2\}\}$.

Lemma 14. *The class $\mathcal{X}_1(i,j,k)$ forms an independent set in $\mathcal{S}_2$.*

Proof. Let $\mathcal{X} = \mathcal{X}_1(i,j,k)$. Observe that for any $X \in \mathcal{X}$ we have $\big\{\{i,j\},\{i,k\}\big\} \subseteq X$ and $\{j,k\} \notin X$. From Remark 2 it easily follows that the relation S_1 satisfies

$$\big\{\{i,j\},\{i,k\}\big\} \subseteq X \subseteq 2^{S_1(x)} \implies x = \{j,k\}.$$

In particular, we get that $X \in \mathcal{X} \implies X \in R_1(\{j,k\})$.

In order to show that $\mathcal{X}$ is an independent set in $\mathcal{S}_2$, let Y and Z be vertices of $\mathcal{S}_2$ such that $Y \in S_2(Z)$. First, if $Y \in \mathcal{X}$, then $Y \in R_1(\{j,k\})$. In particular, this means that $\{j,k\} \in Z$ and we get that $Z \notin \mathcal{X}$. For the second case, suppose $\{j,k\} \notin Z$. This means that a node with colour Z cannot have a successor with colour $\{j,k\}$ in a colouring produced by A_1. Thus, it must be that $Y \notin R_1(\{j,k\})$. By the earlier observation, we get that $Y \notin \mathcal{X}$. □

Lemma 15. *The class $\mathcal{X}_2(i,j,k)$ is independent in $\mathcal{S}_2$.*

Proof. Let $\mathcal{X} = \mathcal{X}_2(i,j,k)$. In this class, for every $X \in \mathcal{X}$ it holds that $\big\{\{i,j\},\{k\}\big\} \subseteq X$. From Remark 2 it follows that

$$\{\{i,j\},\{k\}\} \subseteq X \subseteq 2^{S_1(x)} \implies x \in \{\{i,k\},\{j,k\}\}.$$

Thus, $X \in \mathcal{X} \implies X \in R_1(\{i,k\}) \cup R_1(\{j,k\})$.

In order to show that the class $\mathcal{X}$ forms an independent set in $\mathcal{S}_2$, suppose $Y \in S_2(Z)$ for some $Y, Z \in C_2$. First, if $Y \in \mathcal{X}$, then we have that either $\{i,k\} \in Z$ or $\{j,k\} \in Z$ so $Z \notin \mathcal{X}$. Second, if $Z \in \mathcal{X}$, then $\big\{\{i,k\},\{j,k\}\big\} \cap Z = \emptyset$. This means that a node with colour Z cannot have successor of colour $\{i,k\}$ or $\{j,k\}$ as a successor, hence $Y \notin R_1(\{i,k\}) \cup R_1(\{j,k\})$. □

Lemma 16. *The class $\mathcal{X}_3(i,j,k)$ forms an independent set in $\mathcal{S}_2$.*

Proof. Let $\mathcal{X} = \mathcal{X}_3(i,j,k)$. Observe that for any $X \in \mathcal{X}$ it holds that $\{i,j\} \in X$ and $\big\{\{i,k\},\{j,k\},\{k\}\big\} \cap X = \emptyset$. Using Remark 2 we can check that relation S_1 satisfies

$$X \subseteq 2^{S_1(x)} \implies x = k.$$

Thus, $X \in \mathcal{X} \implies X \in R_1(k)$.

To see that $\mathcal{X}$ is an independent set in $\mathcal{S}_2$, let $Y \in S_1(Z)$. There are two cases to consider. First, if $Y \in \mathcal{X}$, then $Y \in R_1(k)$. That is a node with colour k can output colour Y. Thus, $k \in Z$, so we must have that $Z \notin \mathcal{X}$. Second, if $Z \in \mathcal{X}$, then $k \notin Z$ which means that $Y \notin R_1(k)$. Thus, $Y \notin \mathcal{X}$. □

4.5 Computational Proof of Lemma 13

We now give a *computational proof* of Lemma 13, that is, we show how to easily verify with a computer that the claim holds. Essentially this amounts to checking that for every choice of $A = A_0$, the successor graph $\mathcal{S}_2(A)$ is colourable with 16 colours. However, since any successor graph $\mathcal{S}_2(A)$ depends on the choice of initial one-sided 3-colouring algorithm $A = A_0$, and there are potentially many choices for A, we instead bound the chromatic number of a closely-related graph $\mathcal{S}_2^*$ that contains $\mathcal{S}_2(A)$ for any A as a subgraph.

To construct the graph $\mathcal{S}_2^*$, we consider the successor graph of a "worst-case" algorithm that may output "all possible" colours in its output set. Specifically, this means that we simply replace the subset relation in Remarks 1 and 2 with an equality. Therefore, the graph $\mathcal{S}_2^*$ can be constructed using a fairly straightforward computer program, with a mechanical application of the definitions. The end result is a dense graph on 55 nodes; its complement is shown in Figure 3.

It is now easy to discover a colouring of graph $\mathcal{S}_2^*$ that uses 16 colours with the help of e.g. modern SAT solvers. This implies that any subgraph $\mathcal{S}_2(A)$ can also be coloured with 16 colours and Lemma 13 follows.

5 Main Theorems

We now have all the pieces for proving Theorem 1:

Theorem 1. *For infinitely many values of n, it takes exactly $\frac{1}{2}\log^* n$ rounds to compute a 3-colouring of a directed path.*

Proof. Let $n = {}^{2k+1}2 + 1$ for any $k \geq 2$. Be Lemma 2 we have the identity

$$C(n,3) = \lceil T(n,3)/2 \rceil \tag{1}$$

and from Theorems 4 and 5 we get that

$$2k + 1 \leq T(n,3) \leq 2k + 2,$$

which together with (1) yields $C(n,3) = k + 1$. Since $\log^* n = 2k + 2$ it follows that $C(n,3) = k + 1 = \log^* n/2$. □

For the remaining values of n we get almost-tight bounds. There remains a slack of *one* communication round in the upper and lower bounds for $C(n,3)$.

Theorem 17. *For any $n \geq 4$,*

$$\left\lceil \frac{1}{2}\left(\log^* n - 1\right) \right\rceil \leq C(n,3) \leq \left\lceil \frac{1}{2}\left(\log^* n + 1\right) \right\rceil.$$

Proof. For $n = 4$, we have shown that $T(4,3) = 2$ so the bounds follow. Fix $n > 4$. Now there exists some $h > 1$ such that $n \in \{{}^h2 + 1, \ldots, {}^{h+1}2\}$ and $h = \log^* n - 1$. Theorems 4 and 5 give us the bounds

$$\log^* n - 1 = h \leq T(n,3) \leq h + 2 = \log^* n + 1$$

and since $C(n,3) = \lceil T(n,3)/2 \rceil$, the claimed bounds follow. □

6 Conclusions and Discussion

In this work we gave exact and near-exact bounds on the complexity of distributed graph colouring. The key result is that the complexity of colour reduction from n to 3 on directed paths and cycles is exactly $\frac{1}{2}\log^* n$ rounds for infinitely many values of n, and very close to it for all values of n.

In essence, we have shown that the colour reduction algorithm by Naor and Stockmeyer is near-optimal, while the algorithm by Cole and Vishkin is suboptimal. We have also seen that Linial's lower bound had still some room for improvements.

One of the novel techniques of this work was the use of ***computers in lower-bound proofs***. Two key elements are results of a computer search:

- Lemma 9: The proof of $T(16,3) \geq 3$ is based on the analysis of the chromatic number of the neighbourhood graph $\mathcal{N}_{7,1}$.
- Lemma 10: The proof of $T(n,16) \leq T(n,3) - 2$ is based on the analysis of the chromatic number of the successor graph $\mathcal{S}_2$.

In both cases we used computers to analyse the chromatic numbers of various successor graphs and neighbourhood graphs, in order to find the right parameters for our needs.

The idea of analysing ***neighbourhood graphs*** and their chromatic numbers is commonly used in the context of human-designed lower-bound proofs [7, 10, 13, 14]. It is also fairly straightforward to construct neighbourhood graphs so that we can use computers and graph-colouring algorithms to discover new upper bounds [18], and the same technique can be used to prove lower bounds on $T(n,c)$ for small, fixed values of n and c; in our case we used it to bound $T(16,3)$. However, this does not yield bounds on, e.g., $T(n,3)$ for large values of n.

The key novelty of our work is that we can use the chromatic number of ***successor graphs*** to give improved bounds on $T(n,3)$ for all values of n. To do that, it is sufficient to find a successor graph $\mathcal{S}_k$ with a small chromatic number, apply Lemma 12. The same technique can be also used to study $T(n,c)$ for any fixed $c \geq 3$.

Acknowledgements

We thank Juho Hirvonen for helpful comments. Parts of this work are based on the first author's MSc thesis [18]. Computer resources were provided by the Aalto University School of Science "Science-IT" project, and by the Department of Computer Science at the University of Helsinki.

References

[1] Matti Åstrand and Jukka Suomela. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proc. 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010)*, pages 294–302. ACM Press, 2010. doi:10.1145/1810479.1810533.

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

[3] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[5] Andrzej Czygrinow, Michał Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *Proc. 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008. doi:10.1007/978-3-540-87779-0_6.

[6] Faith E. Fich and Vijaya Ramachandran. Lower bounds for parallel computation on linked structures. In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 1990)*, pages 109–116. ACM Press, 1990. doi:10.1145/97444.97676.

[7] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2007. doi:10.1007/978-3-540-73420-8_22.

[8] Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998. doi:10.1137/S0097539794261118.

[9] Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988. doi:10.1137/0401044.

[10] Fabian Kuhn and Roger Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006)*, pages 7–15. ACM Press, 2006. doi:10.1145/1146381.1146387.

[11] Juhana Laurinharju and Jukka Suomela. Brief announcement: Linial's lower bound made easy. In *Proc. 33rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2014)*, pages 377–378. ACM Press, 2014. doi:10.1145/2611462.2611505. arXiv:1402.2552.

[12] Christoph Lenzen and Boaz Patt-Shamir. Improved distributed Steiner forest construction. In *Proc. 33rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2014)*, pages 262–271. ACM Press, 2014. doi:10.1145/2611462.2611464. arXiv:1405.2011.

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

[14] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991. doi:10.1137/0404036.

[15] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[16] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001. doi:10.1007/PL00008932.

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

[18] Joel Rybicki. Exact bounds for distributed graph colouring. Master's thesis, Department of Computer Science, University of Helsinki, May 2011. `http://urn.fi/URN:NBN:fi-fe201106091715`.

[19] Jukka Suomela. *Distributed Algorithms*. 2014. Online textbook. `http://users.ics.aalto.fi/suomela/da/`.

[20] Roger Wattenhofer. Lecture notes on principles of distributed computing, 2013. `http://dcg.ethz.ch/lectures/podc_allstars/`.