DISS. ETH NO. 19459

# Synchronization and Symmetry Breaking in Distributed Systems

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

Christoph Lenzen

Dipl.-Math., Universität Bonn

born 12.06.1982

citizen of
Germany

accepted on the recommendation of

Professor Roger Wattenhofer, examiner
Professor Danny Dolev, co-examiner
Professor Berthold Vöcking, co-examiner

2010

## Abstract

An emerging characteristic of modern computer systems is that it is becoming ever more frequent that the amount of communication involved in a solution to a given problem is the determining cost factor. In other words, the convenient abstraction of a random access memory machine performing sequential operations does not adequately reflect reality anymore. Rather, a multitude of spatially separated agents cooperates in solving a problem, where at any time each individual agent has a limited view of the entire system's state. As a result, coordinating these agents' efforts in a way making best possible use of the system's resources becomes a fascinating and challenging task. This dissertation treats of several such coordination problems arising in distributed systems.

In the *clock synchronization* problem, devices carry clocks whose times should agree to the best possible degree. As these clocks are not perfect, the devices need to perpetually resynchronize by exchanging messages repeatedly. We consider two different varieties of this problem. First, we examine the problem in sensor networks, where for the purpose of energy conservation it is mandatory to reduce communication to a minimum. We give an algorithm that achieves an asymptotically optimal maximal clock difference throughout the network using a minimal number of transmissions. Subsequently, we explore a worst-case model allowing for arbitrary network dynamics, i.e., network links may fail and (re)appear at arbitrary times. For this model, we devise an algorithm achieving an asymptotically optimal gradient property. That is, if two devices in a larger network have access to precise estimates of each other's clock values, their clock difference is much smaller than the maximal one. Naturally, this property can only hold for devices that had such estimates for a sufficiently long period of time. We prove that the time span necessary for our algorithm to fully establish the gradient property when better estimates become available is also asymptotically optimal.

Many *load balancing* tasks can be abstracted as distributing $n$ balls as evenly as possible into $n$ bins. In a distributed setting, we assume the balls and bins to act as independent entities that seek to coordinate at a minimal communication complexity. We show that under this constraint, a natural class of algorithms requires a small, but non-constant number of communication rounds to achieve a constant maximum bin load. We complement the respective bounds by demonstrating that if any of the preconditions of the lower bound is dropped, a constant-time solution is possible.

Finally, we consider two basic combinatorial structures, *maximal independent sets* and *dominating sets*. A maximal independent set is a subset of the agents containing no pair of agents that can communicate directly, while there is no agent that can be added to the set without destroying this property. A dominating set is a subset of the agents that—as a whole—can contact all agents by direct communication. For several families of graphs, we shed new light on the distributed complexity of computing dominating sets of approximatively minimal size or maximal independent sets, respectively.

## Zusammenfassung

Moderne Computersysteme zeichnen sich in zunehmendem Maße dadurch aus, dass das Kommunikationsvolumen den bestimmenden Kostenfaktor bei der Lösung eines gegebenen Problems darstellt. In der Folge wird die klassische Abstraktion einer random access machine, die sequentielle Operationen ausführt, der Realität heutigen Rechnens nicht mehr gerecht. Vielmehr wird die Lösung durch eine Vielzahl interagierender Systemkomponenten bestimmt, die für sich genommen zu keiner Zeit Zugriff auf den Gesamtzustand des Systems haben. Vor diesem Hintergrund erweist es sich als ebenso fordernde wie fesselnde Aufgabe, die einzelnen Teile des Systems derart zu koordinieren, dass eine optimale Nutzung der verfügbaren Resourcen erreicht wird. In dieser Dissertation behandeln wir verschiedene Koordinationsprobleme, die in verteilten Systemen auftreten.

*Uhrensynchronisation* ist eine Aufgabe, die sich in verteilten Systemen daraus ergibt, dass die lokalen Uhren einzelner Komponenten nicht exakt gleich schnell laufen. Wir behandeln zwei Spielarten dieses Themas. Zunächst untersuchen wir Sensornetzwerke, in denen begrenzte Energiereserven es erfordern, den Funkverkehr auf ein Minimum zu beschränken. Wir beschreiben einen Algorithmus, der unter diesen Bedingungen die maximale Uhrendifferenz im System asymptotisch minimiert. Anschliessend diskutieren wir ein worst-case Modell, in dem das Netzwerk sich beliebig ändert, das heißt Verbindungen zu beliebigen Zeiten ausfallen und aufgebaut werden können. Wir präsentieren einen Algorithmus mit optimaler Gradienteneigenschaft. Dies bedeutet, dass wann immer zwei Teilnehmer in einem grösseren Netzwerk für genügend lange Zeit gegenseitig auf zuverlässige Schätzwerte ihrer Uhrenwerte zugreifen können, die Differenz ihrer Uhrenwerte deutlich kleiner als die maximale im System ist. Unser Algorithmus minimiert asymptotisch die Zeitspanne, die eine Verbindung existieren muss, bis sie der Gradienteneigenschaft genügt.

In vielen Fällen können *Lastverteilungsaufgaben* durch ein abstraktes Modell beschrieben werden, in dem $n$ Bälle $n$ Urnen zugeordnet werden. In einem verteilten System nimmt man dabei an, dass sowohl Bälle als auch Urnen eigenständig operieren. Ziel ist, bei minimaler Kommunikation die maximale Anzahl Bälle in einer Urne konstant zu beschränken. Wir werden zeigen, dass für eine natürliche Klasse von Algorithmen die dafür nötige Anzahl von Kommunikationsrunden zwar langsam wachsend, jedoch nicht unabhängig von $n$ ist. Wir ergänzen dieses Ergebnis durch den Nachweis, dass Fallenlassen einer beliebigen Voraussetzung der entsprechenden unteren Schranke eine Lösung des Problems in konstant vielen Kommunikationsrunden ermöglicht.

Schliesslich untersuchen wir zwei grundlegende kombinatorische Strukturen. Eine *maximale stabile Menge* ist eine nicht vergrösserbare Teilmenge der Komponenten, so dass kein Paar aus dieser Menge direkt kommunizieren kann. Ein *dominierende Menge* ist eine Teilmenge der Komponenten, die zusammengenommen das gesamte System direkt kontaktieren kann. Wir zeigen für verschiedene Graphfamilien Komplexitätsschranken für die Berechnung von maximalen stabilen Mengen beziehungsweise kleinen dominierenden Mengen.

**Acknowledements**

First of all, I would like to express my gratitude to my advisor Roger Wattenhofer. Not only did he guide me during the three years of my graduate studies, also he always tried to understand me even when I clearly was not making any sense. Thanks go to my co-examiners Berthold Vöcking and Danny Dolev for reviewing this thesis and giving valuable feedback. I am indebted to my co-authors Fabian Kuhn, Thomas Locher, Rotem Ohsman, Yvonne-Anne Pignolet, Philipp Sommer, and Jukka Suomela, who contributed in many ways to this work. In addition, Fabian, Thomas, Yvonne-Anne, and Philipp carefully proofread large parts of this thesis and suggested numerous improvements. I am grateful to all current and former collaborators from the Distributed Computing Group for making my stay a pleasant and joyful experience. In particular, I take my hat off to my office mates Raphael Eidenbenz and Tobias Langner (and for reasons named elsewhere also to Yvonne-Anne Pignolet) who created a good working atmosphere while enduring my unconditional monologues. For varying reasons, special thanks go to Keren Censor, Sebastian Daum, Michael Kuhn, Tobias Langner, Topi Musto, Christian Scheideler, Ulrich Schmid, Reto Spöhel, and Jukka Suomela. Finally, I am beholden to all the people that are not named here explicitly, but supported me and this work during my time as Ph.D. student.

# Contents

# Chapter 1

# Introduction

*"Can't you use shorter sentences?" – My girlfriend after reading some random lines from this thesis.*

In large parts, the invention of electronic computing has shaped—and still *is* shaping—our modern society. Traditionally, distributed computing contributed to this process in areas like fault-tolerant computing, sensor networks, and the Internet. Incessantly, it has been of growing importance for day-to-day technology.

Nowadays, this is still true. For one thing, computational power has become incredibly cheap. Today, even the most simple "mobile phone" is in fact a portable computer, faster than processors employed in supercomputers about three decades ago [20]. Arguably, advances in software developing and testing, programming languages, and last but not least basic algorithms had an even greater impact on the capabilities of current standard devices. Considering that these devices get more and more interconnected, be it via the Internet or direct wireless communication, all ingredients of a powerful distributed system are present. This opens the door to a multitude of applications, ranging e.g. over social networking, exchanging and evaluating data, environmental monitoring, and controlling other devices.

It is less noticeable, but maybe even more dramatic, that we are hitting physical barriers preventing to amass ever more *sequential* computing power in a single processor. It becomes increasingly difficult to miniaturize chip components further, making it harder and harder to maintain the illusion of a monolithic system operating in synchronous steps. This motivates hardware vendors, eager to maintain Moore's law, to switch to an exponentially growing number of cores. It is important to understand that this constitutes a fundamental change. In a sequential system, the necessary effort to solve a given task can be concisely expressed in terms of the number of required computational steps. Even in a distributed system where all nodes (i.e., participants) have the same capabilities, one cannot simply divide this measure by the number of these nodes to understand the distributed complexity of a problem. Some tasks are inherently sequential and simply cannot be parallelized. In case parallelism is indeed possible, *communication* becomes an essential part of the process. This communication serves to exchange intermediate results, but also to establish *coordination* among the nodes in the network.

Coordination can be achieved in various ways. Obviously, one distinguished node could manage the system by commanding the others. However, this trivial approach has considerable drawbacks. On the one hand, it requires to collect and process all relevant information in a single spot. This might without need reduce the amount of concurrency achieved, as merely one node does the respective computations. Sometimes, it is actually outright impossible, because no single device has sufficient capabilities. On the other hand, a centralized authority is a single point of failure, throwing away the possibility to perform a given task despite a minority of the individual components failing.

One possible alternative is that each node collects information from other nodes which are "close" in the sense that they can be contacted quickly, and acts according to a scheme avoiding conflicting actions. Depending on the task to solve, such *local algorithms* can be surprisingly efficient. For many of these algorithms, it is imperative to first *break symmetry* in order to avoid conflicting or redundant actions, which otherwise would thwart progress or waste resources. Moreover, typically nodes need to *synchronize* their actions. This can be done explicitly by message exchange, or implicitly by means of timing information.

In this thesis, we will investigate a number of such basic distributed coordination tasks. We will present and analyze primitives for *clock synchronization* (Part I), randomized *load balancing* (Part II), and *graph problems* on restricted families of graphs (Part III). Our main goal is to extend the knowledge on the fundamental limits to the degree of concurrency and efficiency at which these problems can be solved. We aim at a mathematically rigid assessment of the distributed complexity of our algorithms as well as the amount of resources that must be used by *any* algorithm for the respective task. This demands abstract models, which however still must capture the crucial properties of the considered system. We hope to have succeeded in the tightrope walk between oversimplification and getting lost in details, obtaining clear theoretical statements that are meaningful in practice.

# Chapter 2

# Preliminaries

*"Distributed computing? Shouldn't be that different from ordinary computing, right?" – Synopsis of my knowledge on distributed computing at the time when I began my graduate studies.*

In this chapter, we summarize basic notation and some well-known results we will rely on. We will not prove the given statements; the goal of this chapter is to provide a reference in order to avoid lack of clarity in subsequent chapters. Consequently, the reader is encouraged to quickly review the notation, skip the lemmas and theorems, and come back to this chapter if required later on.

## 2.1 Basic Model and Notation

By $\mathbb{N}$ we denote the set of natural numbers and by $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$ the natural numbers together with 0. Similarly, $\mathbb{R}$ denotes the Reals, $\mathbb{R}^+ := \{x \in \mathbb{R} \,|\, x > 0\}$ the strictly positive Reals, and $\mathbb{R}_0^+ := \{x \in \mathbb{R} \,|\, x \geq 0\}$ the non-negative Reals. We will use Landau notation with respect to the asymptotics towards $+\infty$, i.e., according to the following definitions.

**Definition 2.1** (Landau Symbols). *Given $f, g : A \to \mathbb{R}_0^+$, where $A \subseteq \mathbb{R}$ with* $\sup A = \infty$*, we define*

$$\begin{aligned}
f \in \mathcal{O}(g) &\quad\Leftrightarrow\quad \exists C_1, C_2 \in \mathbb{R}_0^+\ \forall x \in A:\ f(x) \leq C_1 g(x) + C_2\\
f \in o(g) &\quad\Leftrightarrow\quad \lim_{\substack{C\in\mathbb{R}_0^+\\ C\to\infty}} \sup_{\substack{x\in A\\ x\geq C}} \left\{\frac{f(x)}{g(x)}\right\} = 0\\
f \in \Omega(g) &\quad\Leftrightarrow\quad \exists C_1, C_2 \in \mathbb{R}_0^+\ \forall x \in A:\ C_1 f(x) + C_2 \geq g(x)\\
f \in \omega(g) &\quad\Leftrightarrow\quad \lim_{\substack{C\in\mathbb{R}_0^+\\ C\to\infty}} \sup_{\substack{x\in A\\ x\geq C}} \left\{\frac{g(x)}{f(x)}\right\} = 0\\
f \in \Theta(g) &\quad\Leftrightarrow\quad f \in \mathcal{O}(g) \cap \Omega(g).
\end{aligned}$$

**Definition 2.2** (Logarithms and Polylogarithmic Bounds). *For $x \in \mathbb{R}^+$, by $\log x$ and $\ln x$ we denote the logarithms to base 2 and e, respectively, where $e = \lim_{x\to\infty}(1+1/x)^x$ is Euler's number. We define $\log^{(i)} x$ (for feasible values of $x$) to be the $i \in \mathbb{N}$ times iterated logarithm, whereas $\log^r x := (\log x)^r$ for any $r \in \mathbb{R}^+$. We say that the function $f(x) \in \operatorname{polylog} x$ if $f(x) \in \mathcal{O}(\log^C x)$ for a constant $C \in \mathbb{R}^+$.*

**Definition 2.3** (Tetration and $\log^*$). *For $k \in \mathbb{N}$ and $b \in \mathbb{R}^+$, the $k^{th}$* tetration *of $b$ is given by*

$$^k b := b^{b^{\cdot^{\cdot^{\cdot^{b}}}}} \Big\} k \text{ times.}$$

*For $x \in \mathbb{R}^+$, we define $\log^* x$ recursively by*

$$\log^* x := \begin{cases} 1 + \log^* \log x & \text{if } x > 1\\ 0 & \text{otherwise.}\end{cases}$$

*In particular, $\log^*\left({}^k 2\right) = k$ for all $k \in \mathbb{N}$.*

Throughout this thesis, we will describe distributed systems according to the standard message passing model. The network will be modelled by a simple graph $G = (V, E)$, where $V$ is the set of nodes and $\{v, w\} \in E$ means that $v$ and $w$ share a bidirectional communication link. We will employ the following basic notation.

**Definition 2.4** (Paths, Distances, and Diameter). *Given a graph $G = (V, E)$, a path of length $k \in \mathbb{N}$ is a sequence of nodes $(v_0, \ldots, v_k) \in V$ such that $\{v_i, v_{i-1}\} \in E$ for all $i \in \{1, \ldots, k\}$. The distance $d(v, w)$ of two nodes $v, w \in V$ is the length of a shortest path between $v$ and $w$. The diameter $D$ of $G$ is the maximum distance between any two nodes in the graph.*

**Definition 2.5** (Neighborhoods). *Given the graph $G = (V, E)$, we define*

- *the (exclusive)* neighborhood *$\mathcal{N}_v := \{w \in V \mid \{v, w\} \in E\}$ of node $v \in V$,*

- *the* degree $\delta_v := |\mathcal{N}_v|$ *of* $v \in V$,
- *the* maximum degree $\Delta := \max_{v \in V}\{\delta_v\}$,
- *for* $k \in \mathbb{N}$ *the (inclusive)* $k$-neighborhood $\mathcal{N}_v^{(k)} := \{w \in V \,|\, d(v,w) \leq k\}$ *of* $v \in V$,
- *and the (inclusive)* neighborhood $\mathcal{N}_A^+ := \bigcup_{v \in A} \mathcal{N}_v^{(1)}$ *of set* $A \subseteq V$.

*To facilitate intuition, we will denote the inclusive* 1*-neighborhood of node* $v \in V$ *by* $\mathcal{N}_v^+ := \mathcal{N}_v^{(1)}$.

In bounded-delay networks (which are considered in Part I of this thesis), nodes react to *events*, which are triggered by receiving messages or reaching a (previously defined) value on a local clock. When an event is triggered, a node may perform local computations, send messages that will be received within bounded time, and define future local times at which events will be triggered locally. These actions take no time; in case two events are triggered at a node at the same time, they are ordered arbitrarily and processed sequentially. We will use events triggered by the local clock implicitly, as we employ a high-level description of our algorithms. We point out, however, that one can translate all algorithms into this framework. The state of each node is thus a function of the *real time* $t \in \mathbb{R}_0^+$. If at time $t$ the state of a variable (function, etc.) $x$ changes instantaneously, we define $x(t)$ to be the value after this change has been applied.

In contrast, in Parts II and III of our exposition we employ a synchronous model, where computation advances in rounds. In each round, nodes send messages, receive messages sent by their neighbors, and perform local computations. The state of a node thus becomes a function of the current round $r \in \mathbb{N}$.

Despite aiming for simple algorithms, we do not impose any constraints on nodes' memory and the local computations they may perform. Note, however, that one should avoid techniques like e.g. collecting the whole topology of neighborhood up to a certain distance and subsequently solve NP-hard problems locally.

## 2.2 Standard Definitions and Tools

### Probabilistic Tools

All random variables in this thesis will be real-valued, hence we will not repeat this in our statements. We denote the expectation and variance of random variable $X$ by $\mathbb{E}[X]$ and $\mathrm{Var}[X]$, respectively.

**Theorem 2.6** (Markov's Bound)**.** *For any random variable $X$ and any $C \in \mathbb{R}^+$, it holds that*

$$P[|X| \geq C] \leq \frac{\mathbb{E}[X]}{C}.$$

When deriving probabilistic bounds, we will strive for results that are not certain, but almost guaranteed to hold.

**Definition 2.7** (With High Probability)**.** *A stochastic event $\mathcal{E}(c)$, where $c \in \mathbb{R}^+$ is arbitrary, is said to occur* with high probability (w.h.p.)*, if $P[\mathcal{E}(c)] \geq 1 - 1/n^c$. Throughout this thesis, we will use $c$ with this meaning only and will therefore not define it again. When it comes to Landau notation, $c$ is treated as a constant, e.g. the values $C_1$ and $C_2$ from the definition of $\mathcal{O}(\cdot)$ may depend on $c$.*

The advantage of this definition lies in its transitivity, as for instance the statements "Each node completes phase $i$ of the algorithm in $\mathcal{O}(\log n)$ rounds w.h.p.", where $i \in \{1, \dots, \mathcal{O}(\log n)\}$, imply the statement "All phases complete in $\mathcal{O}(\log^2 n)$ rounds w.h.p." Formally, the following lemma holds.

**Lemma 2.8.** *Assume that events $\mathcal{E}_i(c)$, $i \in \{1, \dots, N\}$, occur w.h.p., where $N \leq n^C$ for some constant $C \in \mathbb{R}^+$. Then event $\mathcal{E}(c) := \bigwedge_{i=1}^{N} \mathcal{E}_i(\tilde{c})$ occurs w.h.p., where $\tilde{c} := c + C$.*

*Proof.* The $\mathcal{E}_i$ occur w.h.p., so for any value $c \in \mathbb{R}^+$ we may choose $\tilde{c} := c+C \in \mathbb{R}^+$ and have $P[\mathcal{E}_i(\tilde{c})] \geq 1-1/n^{\tilde{c}} \geq 1-1/(Nn^c)$ for all $i \in \{1, \dots, N\}$. By the union bound this implies $P[\mathcal{E}(c)] \geq 1 - \sum_{i=1}^{N} P[\overline{\mathcal{E}_i}] \geq 1 - 1/n^c$. □

We will not invoke this lemma explicitly in our proofs. The purpose of this statement rather is to demonstrate that any number of asymptotic statements holding w.h.p. that is polynomial in $n$ is also *jointly* true w.h.p., regardless of dependencies. With this in mind, we will make frequent implicit use of this lemma.

**Definition 2.9** (Uniformity and Independence)**.** *A discrete random variable is called* uniform*, if all its possible outcomes are equally likely. Two random variables $X_1$ and $X_2$ are* independent*, if $P[X_1 = x_1] = P[X_1 = x_1 | X_2 = x_2]$ for any two $x_1, x_2 \in \mathbb{R}$ (and vice versa). A set $\{X_1, \dots, X_N\}$ of random variables is independent if, for all $i \in \{1, \dots, N\}$, $X_i$ is independent from $(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_N)$, i.e., the tuple listing the outcomes of all $X_j \neq X_i$. The set $\{X_1, \dots, X_N\}$ is* uniformly and independently random (u.i.r.) *if it is independent and consists of uniform random variables. Two sets of random variables $X = \{X_1, \dots, X_N\}$ and $Y = \{Y_1, \dots, Y_M\}$ are independent of each other if all $X_i \in X$ are independent from $(Y_1, \dots, Y_M)$ and all $Y_j \in Y$ are independent from $(X_1, \dots, X_N)$.*

Frequently w.h.p. results are deduced from Chernoff bounds, which provide exponential probability bounds regarding sums of Bernoulli variables (which are either one or zero). Common formulations assume independence of these variables, but the following more general condition is sufficient.

**Definition 2.10** (Negative Association)**.** *The set of random variables $X_i$, $i \in \{1, \dots, N\}$, is* negatively associated *if and only if for all disjoint subsets $I, J \subseteq \{1, \dots, N\}$ and all functions $f : \mathbb{R}^{|I|} \to \mathbb{R}$ and $g : \mathbb{R}^{|J|} \to \mathbb{R}$ that are either increasing in all components or decreasing in all components we have*

$$\mathbb{E}[f((X_i)_{i\in I}) \cdot g((X_j)_{j\in J})] \leq \mathbb{E}[f((X_i)_{i\in I})] \cdot \mathbb{E}[g((X_j)_{j\in J})].$$

Note that independence trivially implies negative association, but not vice versa.

**Theorem 2.11** (Chernoff's Bound: Upper Tail)**.** *Given negatively associated Bernoulli variables $X_1, \dots, X_N$, define $X := \sum_{i=1}^{N} X_i$. Then for any $\delta \in \mathbb{R}^+$, we have that*

$$P[X > (1+\delta)\mathbb{E}[X]] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathbb{E}[X]}.$$

**Theorem 2.12** (Chernoff's Bound: Lower Tail)**.** *Given negatively associated Bernoulli variables $X_1, \dots, X_N$, define $X := \sum_{i=1}^{N} X_i$. Then for any $\delta \in (0, 1]$, it holds that*

$$P[X < (1-\delta)\mathbb{E}[X]] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mathbb{E}[X]}.$$

**Corollary 2.13.** *For negatively associated Bernoulli variables $X_1, \dots, X_N$, define $X := \sum_{i=1}^{N} X_i$. Then*

*(i)* $X \in \mathbb{E}[X] + \mathcal{O}\left(\log n + \sqrt{\mathbb{E}[X]\log n}\right)$ *w.h.p.*

*(ii)* $\mathbb{E}[X] \in \mathcal{O}(1) \Rightarrow X \in \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ *w.h.p.*

*(iii)* $\mathbb{E}[X] \in \mathcal{O}\left(\frac{1}{\sqrt{\log n}}\right) \Rightarrow X \in \mathcal{O}\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ *w.h.p.*

*(iv)* $P[X = 0] \leq e^{-\mathbb{E}[X]/2}$

*(v)* $\mathbb{E}[X] \geq 8c\log n \Rightarrow X \in \Theta(\mathbb{E}[X])$ *w.h.p.*

*(vi)* $\mathbb{E}[X] \in \omega(\log n) \Rightarrow X \in (1 \pm o(1))\mathbb{E}[X]$ *w.h.p.*

We need a means to show that random variables are negatively associated.

**Lemma 2.14.**

*(i)* *If $X_1,\dots,X_N$ are Bernoulli variables satisfying $\sum_{i=1}^N X_i = 1$, then $\{X_1,\dots,X_N\}$ is negatively associated.*

*(ii)* *Assume that $X$ and $Y$ are negatively associated sets of random variables, and that $X$ and $Y$ are mutually independent. Then $X \cup Y$ is negatively associated.*

*(iii)* *Suppose $\{X_1,\dots,X_N\}$ is negatively associated. Given $I_1,\dots,I_k \subseteq \{1,\dots,N\}$, $k \in \mathbb{N}$, and functions $h_j : \mathbb{R}^{|I_j|} \to \mathbb{R}$, $j \in \{1,\dots,k\}$, that are either all increasing or all decreasing, define $Y_j := h_j((X_i)_{i\in I_j})$. Then $\{Y_1,\dots,Y_k\}$ is negatively associated.*

This lemma and Corollary 2.13 imply strong bounds on the outcome of the well-known balls-into-bins experiment.

**Lemma 2.15.** *Consider the random experiment of throwing $M$ balls u.i.r. into $N$ bins. Denote by $Y^k = \left\{Y_i^k\right\}_{i\in\{1,\dots,N\}}$ the set of Bernoulli variables being 1 if and only if at least (at most) $k \in \mathbb{N}_0$ balls end up in bin $i \in \{1,\dots,N\}$. Then, for any $k$, $Y^k$ is negatively associated.*

The following special case will prove to be helpful.

**Corollary 2.16.** *Throw $M \leq N\ln N/(2\ln\ln n)$ balls u.i.r. into $N$ bins. Then $(1\pm o(1))Ne^{-M/N}$ bins remain empty w.h.p.*

Another inequality that yields exponentially falling probability bounds is typically referred to as Azuma's inequality.

**Theorem 2.17** (Azuma's Inequality)**.** *Assume that $X$ is a random variable that is a function of independent random variables $X_1,\dots,X_N$. Assume that changing the value of a single $X_i$ for some $i \in \{1,\dots,N\}$ changes the outcome of $X$ by at most $\delta_i \in \mathbb{R}^+$. Then for any $t \in \mathbb{R}_0^+$ we have*

$$P[|X - \mathbb{E}[X]| > t] \quad \leq \quad 2e^{-\frac{t^2}{2\sum_{i=1}^N \delta_i^2}}.$$

## Normally Distributed Random Variables

**Definition 2.18** (Normal Distribution)**.** *The random variable $X$ is* normally distributed *if its density function is the bell curve*

$$f(x) = \frac{1}{\sqrt{2\pi\,\mathrm{Var}[X]}}e^{-\frac{(x-\mathbb{E}[X])^2}{2\,\mathrm{Var}[X]}}.$$

Sums of normally distributed variables are again normally distributed.

**Lemma 2.19.** *Given normally distributed random variables $X_1, \dots, X_N$, their sum $X := \sum_{i=1}^N X_i$ is normally distributed with expectation $\mathbb{E}[X] = \sum_{i=1}^N \mathbb{E}[X_i]$ and variance $\mathrm{Var}[X] = \sum_{i=1}^N \mathrm{Var}[X_i]$.*

For our purposes, normally distributed random variables exhibit a very convenient behaviour.

**Lemma 2.20.** *For any given normally distributed random variable $X$, we have that*

$$P\left[|X - \mathbb{E}[X]| > \sqrt{\mathrm{Var}[X]}\right] \in \Omega(1),$$

*i.e., the probability to deviate by more than one standard deviation is constant, whereas*

$$P\left[|X - \mathbb{E}[X]| \leq \delta\sqrt{\mathrm{Var}[X]}\right] \in 1 - e^{-\Omega(\delta^2 \log \delta)}$$

*for any $\delta \in \mathbb{R}^+$.*

## Simple Linear Regression

**Definition 2.21** (Simple Linear Regression)**.** *Given data points $(x_i, y_i)$, $i \in \{1, \dots, N\}$, such that not all $x_i$ are the same, their* linear regression *is the line*

$$\hat{f}(x) = \hat{s}x + \hat{t},$$

*where $\hat{s}, \hat{t} \in \mathbb{R}$ are minimizing the expression*

$$\sum_{i=1}^N \left(\hat{f}(x_i) - y_i\right)^2.$$

*Denoting by $\bar{\cdot}$ the average of the respective values $\cdot_i$, $i \in \{1, \dots, N\}$, we have*

$$\begin{aligned} \hat{s} &= \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \overline{x}^2} \\ \hat{t} &= \overline{y} - \hat{s}\,\overline{x}. \end{aligned}$$

Using linear regression on a set of measurements of a linear relation that is inflicted with errors, one can significantly reduce the overall deviation of the estimated line from the true one.

**Theorem 2.22.** *Assume that we are given a set of measurements $(x_i, \hat{y}_i)$ of data points $(x_i, y_i)$, $i \in \{1, \dots, N\}$, obeying the relation $f(x_i) = y_i$, where $f(x) = sx + t$. Furthermore, assume that*

$$\hat{y}_i = y_i + X_i,$$

*where the $X_i$ are identically and independently normally distributed random variables with expectation $\mu$ and variance $\sigma^2$. Denote by $\hat{f}(x) = \hat{s}x + \hat{t}$ the linear regression of the data set $\{(x_i, \hat{y}_i)\}_{i \in \{1,\dots,N\}}$. Then we have that* *(i)* $\hat{s}$ *is normally distributed with* $\mathbb{E}[\hat{s}] = s$ *and* $\operatorname{Var}[\hat{s}] = \sigma^2 / \sum_{i=1}^{N}(x_i - \bar{x})^2$,

*(ii)* $\hat{f}(\bar{x}) - f(\bar{x})$ *is normally distributed with mean* $\mu$ *and variance* $\sigma^2/N$.

## Miscellaneous

In Chapter 5 we will exploit the fact that the maximum of functions which increase at a bounded rate does not grow faster than the maximum of the respective bounds.

**Theorem 2.23.** *Suppose* $f_1, \ldots, f_k : \mathcal{T} \to \mathbb{R}$ *are functions that are differentiable at all but countably many points, where* $\mathcal{T} \subseteq \mathbb{R}$. *Then* $f := \max\{f_1, \ldots, f_k\} : \mathcal{T} \to \mathbb{R}$ *is differentiable at all but countably many points, and it holds for all* $t \in \mathcal{T}$ *for which all involved derivatives exist that*

$$\frac{d}{dt} f(t) = \max_{\substack{i \in \{1,\ldots,k\} \\ f_i(t) = f(t)}} \left\{ \frac{d}{dt} f_i(t) \right\}.$$

In Chapter 13 we will need the following basic statements about planar graphs.

**Lemma 2.24.** *A minor of a planar graph is planar. A planar graph of* $n \geq 3$ *nodes has at most* $3n - 6$ *edges.*

A basic combinatorial structure that will be briefly mentioned in Part III is a node coloring.

**Definition 2.25** (Node Coloring). *A* node coloring with $k \in \mathbb{N}$ colors *is a mapping* $\mathcal{C} : V \to \{1, \ldots, k\}$ *such that no two neighbors have the same color, i.e.,* $\{v, w\} \in E \Rightarrow \mathcal{C}(v) \neq \mathcal{C}(w)$.

# Part I

# Clock Synchronization

# Chapter 3

# An Introduction to Clock Synchronization

*"I believed the topic was dead." – Christian Scheideler's opening to a question concerning a talk about clock synchronization.*

In distributed systems, many tasks rely on—or are simplified by—a common notion of time throughout the system. Globally coherent local times allow for implicit synchronization of the actions of distant devices [12] or chronological ordering of events occurring at distinct nodes. If time is not abstract, but to be understood in the physical sense as provided by e.g. a watch or a system clock, this clears the path for numerous further applications. For instance, the precision up to which an acoustic event can be located by a group of adjacent sensor nodes crucially depends on the exact times when the sensors detect its sound waves.

This distinction between "abstract" and "physical" time is decisive. The goal of synchronizing a distributed system to the degree that nodes have access to a common round counter is addressed by so-called synchronizers [3] and ordering events within the system has been—by and large—understood already in the early days of distributed computing [57]. Having a physically meaningful clock is more demanding in that it requires not only consistent clock values throughout the distributed system, but also clearly defined progress speeds of clocks. This is for instance important when a trajectory is to be (re)constructed out of sensor readings: If clock speeds are arbitrary, the velocity of the observed target cannot be determined accurately. Putting it simply, a second should last about a second, not between zero and ten seconds. If one does *not* care about the progress speed of clocks, *clock skew*, i.e., difference between clock values, can easily be kept small, as one can slow down clocks until stragglers catch up whenever necessary.

But what makes it difficult to prevent that clock skews *arise* if clocks must make progress? To begin with, there is a wide range of scenarios in which it is infeasible that all participants of the system directly access a sufficiently precise source of timing information. Sensor nodes, for instance, can be equipped with GPS receivers, but this might be prohibitively expensive in terms of energy consumption or the network could be indoors. Giving another example, signal propagation speed on computer chips depends on many uncontrollable factors like (local) temperature, variations in quality of components, or fluctuations in supply voltage. Thus, a canonical approach is to equip the participants of the system with their own clocks, which however will exhibit different and varying *clock drifts* for very much the same reasons. Depending on the desired quality of synchronization, it may take more or less time until the clock skew that builds up over time becomes critical. In any case, eventually the devices must communicate in order to adjust their clocks. At this point another obstacle comes into play: the time it takes to transmit a message and process it at the target node can neither be predicted nor be measured precisely. Even if this would be the case, this obstacle could not be overcome completely. Within the time it takes to communicate a value, the clock value of the sender increases by an amount that cannot be determined by the receiver exactly. Thus, nodes suffer from *uncertainty* about neighbors' clock values, and even more so about clock values of remote nodes.

In this thesis, we examine two different models of clock synchronization. The first one is tailored to represent the main characteristics of wireless sensor networks with regard to the clock synchronization problem. In this context, we assume the system to behave comparatively benign. Clock drifts do not change quickly with respect to the time frame relevant for the respective algorithm and are thus kept constant for analysis purposes. The fluctuations in message transmission times are random and independent between transmissions. Although abstracting away from the peculiarities of wireless communication, our theoretical insights are supported by test results from an implementation of *PulseSync*, the algorithm we propose in Section 4.4.

As frequently is the case with clock synchronization, our results reveal that the precise model matters a lot. Denoting by $D$ the diameter of the communication network, we prove a tight probabilistic bound of $\Theta(\sqrt{D})$ on the *global skew* of PulseSync, i.e., the maximum clock skew between any pair of nodes in the system. In contrast, traditional worst-case analysis yields a lower bound of $\Omega(D)$ on the global skew [16].

In Chapter 5 we examine a worst-case model, where clock drifts and uncertainties may vary arbitrarily within possibly unknown bounds. Moreover, we consider dynamic graphs, where the edges of the graph appear and disappear in a worst-case manner. Thus, any upper bound in this model is highly robust, being resilient to anything but maliciously behaving ("Byzantine") nodes. Note that in a system with Byzantine faults, it is mandatory to make sure that an erroneously behaving node cannot pollute the state of others. Obviously, this is impossible if a Byzantine node controls all communication between two parts of a graph. This observation shows that in a Byzantine environment the problem is tied to the topology much stronger and requires more complicated algorithms. For these reasons, Byzantine faults are beyond the scope of our work. To the best of our knowledge, so far the literature has been concerned with Byzantine fault-tolerant clock synchronization algorithms under the assumption of full connectivity only [99]. Even then, the problem of achieving both Byzantine fault tolerance and self-stabilization [26] (see Definition 5.27 and Corollary 5.28) is intricate [11, 27, 28, 35].

It is not difficult to show that the best possible worst-case guarantee on the global skew is linear in the (possibly dynamic) diameter [16, 52, 99]. More surprisingly, even if the graph is static, it is impossible to ensure that the *local skew*, the maximum skew between neighbors, satisfies a bound that is independent of the network diameter [33, 60, 79]. This is of significant interest, as in fact many applications do not necessitate good global synchronization, but merely rely on guarantees on the local skew. For instance, for the aforementioned purpose of acoustic localization we need that nodes that are close to a specific event have closely related clock values. Naturally, these physically clustered nodes will be communicating with each other via a small number of hops. Similarly, *time division multiple access* protocols, where a common channel is accessed by the sharing devices according to a mutually exclusive assignment of time slots, depend on the respective devices having tightly synchronized clocks. Alongside the primary designation of the channel, it can be used to directly exchange timing information between the devices using it. Hence, an efficient utilization of the channel can be achieved provided that the local skew is kept small.

We will show that in any graph, an optimal bound on the local skew on the edges that have been continuously present for a sufficiently long period of time can be maintained by a simple algorithm. This bound is logarithmic in $D$ with a large base of the logarithm, implying that even if the global skew is large, applications depending on the local skew can exhibit a good worst-case scaling behaviour. Moreover, for the proposed algorithm the *stabilization time*, i.e., the time until the strong local skew bounds apply to a newly formed edge, is linear in the bound on the global skew, which is also asymptotically optimal. Remarkably, the *stable local skew* achieved in the subgraph induced by the edges that have been operational without interruption for this time period is almost identical to the local skew that can be guaranteed in a static graph where nodes and edges never fail.

# Chapter 4

# Clock Synchronization in Wireless Networks

*"All the time you said trees are bad. Now, all of a sudden, you want me to change the entire implementation to a tree protocol?"*
*– Philipp Sommer's response to my first sketch of PulseSync.*

In the last two decades, a lot of research has been dedicated to wireless networks. Since such networks do not require a fixed wiring, they are easy to deploy and can be formed on-the-fly when there is need for cooperation between otherwise unrelated mobile devices. On the downside, wireless communication suffers from interference, complicating information exchange between the participants of the system. The fact that energy is typically a scarce resource in wireless networks aggravates this issue further, as one wants to minimize radio usage. In this chapter, we examine the clock synchronization problem in this particular setting. The presented material is based on work co-authored by Philipp Sommer [63].

## 4.1 Model

In a wireless network, communication takes place by radio. In theory, in order to send a message, a node powers up its radio, transmits the message, and powers the radio down. In practice, of course, there are a number of issues. Does the receiver listen on the respective channel? Is there interference with other transmissions? Is an acknowledgement to be sent? If so, was it successfully received, etc. We will not delve into these matters, although

Table 4.1: Energy consumption of radios used in common sensor nodes. Active radios drain by roughly 5 orders of magnitude more power than sleeping devices. The vendors' terms for the mode denoted by "sleep" differ.

| sensor node | transmit [mA] | power [dBm] | listen [mA] | sleep [$\mu$A] |
|---|---|---|---|---|
| Mica2 | 16.5 | 0 | 9.6 | 0.2 |
| Tmote Sky | 17.4 | 0 | 18.8 | 0.02 |
| Crossb. IRIS | 14.5 | 1 | 15.5 | 0.02 |
| TinyNode | 33 | 5 | 14 | 0.2 |

one has to keep the peculiarities of wireless communication in mind when devising clock synchronization protocols for such systems.

Having said this, we choose a simplistic description of the network as a static graph $G = (V, E)$, where $V$ is the set of nodes and $E$ is the set of bidirectional, reliable communication links. If node $v \in V$ sends a message, all neighbors $w \in \mathcal{N}_v$ listening on the channel can receive this message. We focus on the following aspects of wireless systems:

- *Communication is expensive.* The energy consumption of a node whose radio is powered on is orders of magnitude larger than that of a sleeping node (cf. Table 4.1).[1] In fact, in many cases radio usage determines the life-time of a sensor node. Therefore, we want to minimize the amount of communication dedicated to the synchronization routine. Consequently, we require that nodes send and receive (on average) one message per *beacon interval* $B$ only.

- *Communication is inexact.* As mentioned before, it is not possible to learn the exact clock values of communication partners. In the wireless setting, this is mainly due to two causes. Firstly, transmission times vary. This effect can be significantly reduced by MAC layer time-stamping [77], yet a fraction of the transmission time cannot be determined exactly. Secondly, the resolution of the sensor nodes' clocks is limited. Thus, rounding errors are introduced that make it impossible to determine the time of arrival of a message precisely (this can also be improved [97]). As these fluctuations are typically not related between different messages, we model them as independently distributed

[1]Mica 2, Texas Instruments CC1000, focus.ti.com/lit/ds/symlink/cc1000.pdf
Tmote Sky, Texas Instruments CC2420, focus.ti.com/lit/ds/symlink/cc2420.pdf
Crossbow IRIS, Atmel AT86RF230,
atmel.com/dyn/resources/prod_documents/doc5131.pdf
TinyNode, Semtech XE1205, semtech.com/images/datasheet/xe1205.pdf

random variables. For the sake of our analysis, we assume their distributions to be identical and refer to the respective standard deviation as *jitter* $\mathcal{J}$. Our results hold for most "reasonable" distributions. For simplicity, we will however assume normally distributed variables with zero mean in this thesis, a hypothesis which is supported by empirical study [30].

- *Sending times are constrained.* We discussed that in wireless networks one cannot simply send a message whenever it is convenient. In order to account for this, we define the time it takes in each beacon interval between a node receiving and sending a message to be predefined and immutable by the algorithm. For the reason that every node will receive and transmit only once during every interval, we need only a single value $\tau_v \in \mathbb{R}^+$ for each node $v \in V$, denoting the time difference between receiving and sending the respective messages. This time span also accounts for the fact that it takes some time to receive, send, and process messages. Note that this is a simplification in that this time span is variable for several reasons. However, the respective fluctuations are small enough to have negligible effects, as in a real system the fact that radios are powered down most of the time necessitates that nodes can predict when the next message arrives in order to activate the receiver and listen on the appropriate channel.

- *Message size is constrained.* The number of bits in radio messages should be small for various reasons. This is addressed by our algorithm in that the "payload" of a message consists of a small (constant) number of values. We do not formalize this in our model; in particular, we assume unbounded clocks. In practice, a limited number of bits is used to represent clock values and a wrap-around is implemented.

- *Dynamics.* Which nodes can communicate directly may depend on various environmental conditions, in particular interference from inside or outside the network. Thus, in contrast to the previous definition of $G$, the communication graph is typically not static. Moreover, the speed of the nodes' clocks will vary, primarily due to changes in the nodes' temperatures (see Figure 4.1; we remark that nodes equipped with temperature sensors can significantly reduce this influence [97]). We do not capture these aspects in our model, which assumes a static configuration of the system, both with regard to communication and clock rates. This aspect is addressed by the design of the proposed algorithm, which strives for dependency of computed clock values on a short period of time. Thus, the algorithm will adapt fast to changes in topology or clock speeds.

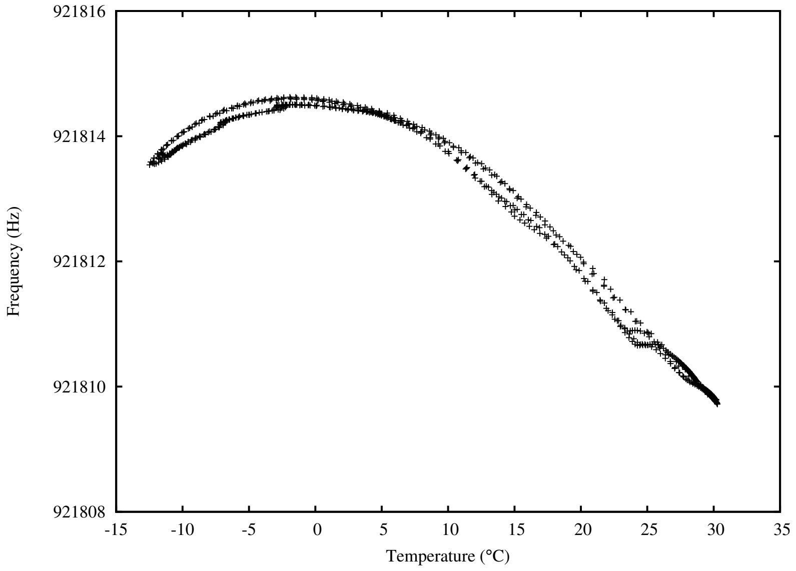

Figure 4.1: Hardware clock frequency of a Mica2 sensor node for different ambient temperatures. A difference of five degrees alters the clock speed by up to one microsecond per second.

Let us now formalize the clock synchronization problem in this communication model. Each node $v \in V$ has a local *hardware clock* $H_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$. It is an affine linear function

$$H_v(t) = o_v + h_v \cdot t,$$

where $o_v \in \mathbb{R}_0^+$ is the *offset* and $h_v$ is the *rate* of $v$'s clock. Node $v$ has access to $H_v(t)$ only, i.e., it can read its local clock value, but does neither know $o_v$ nor $h_v$. The rate $h_v$ determines by how much a local measurement of a difference between two points in time deviates from the correct value. We require that the *relative drift* of $H_v$ is bounded, i.e.,

$$\rho_v := |h_v - 1| \leq \rho < 1.$$

Here $\rho$ is independent of the number of nodes $n$, meaning that each clock progresses at most by a constant factor slower or faster than real time. Typical hardware clocks in sensor nodes exhibit drifts of at most 50 ppm, i.e., $\rho \leq 5 \cdot 10^{-5}$.

Observe that given an infinite number of messages, two neighboring nodes could estimate each other's clock values arbitrarily well. Sending clock updates repeatedly and exploiting independence of message jitters, a node $v \in V$ can approximate the function $H_w$, $w \in \mathcal{N}_v$, arbitrarily precisely in terms of $H_v$ with probability arbitrarily close to 1. For theory, it is thus mainly interesting to study local clocks with fixed drift in combination with algorithms whose output at time $t$ depends on a bounded number of messages only. In light of our previous statements, the same follows canonically from our goals to (*i*) minimize the number of messages nodes send in a given time period and (*ii*) enable the algorithm to deal with dynamics by making it oblivious to information that might be outdated. If we relied on clock values from a large period of time (where the meaning of "large" depends on the speed of changes in environmental conditions), the assumption of clock drifts being constant (up to negligible errors) would become invalid.

A *clock synchronization algorithm* is now asked to derive at each node $v \in V$ a *logical clock* $L_v := \mathbb{R}_0^+ \to \mathbb{R}_0^+$ based on local computations, its hardware clock readings, and the messages exchanged. The algorithm strives to minimize the *global skew*

$$\mathcal{G}(t) := \max_{v,w \in V} \{|L_v(t) - L_w(t)|\}$$

at any time $t$, using few messages and only recent information.

Observe that so far a trivial solution would be to simply set $L_v(t) := 0$ for all times $t$ and all nodes $v \in V$. As mentioned in the introduction, this is not desired as we want $L_v$ to behave like a "real" clock. In particular, we expect clock speeds to be close to one and clock values to be closely related to real time. To avoid a cluttered notation, in this chapter we will adopt the following convention. There is a distinguished root node $r \in V$ that has a perfect clock, i.e., $H_r(t) = t = L_r(t)$ at all times $t$, and nodes try to synchronize their logical clocks with $L_r$. This is known as *external synchronization* in the literature, as opposed to the closely related concept of *internal synchronization* that we will consider in Chapter 5. Observe that

$$\max_{v \in V}\{|L_v(t) - L_r(t)|\} \leq \mathcal{G}(t) \leq 2\max_{v \in V}\{|L_v(t) - L_r(t)|\},$$

i.e., minimizing the global skew is essentially equivalent to synchronizing clocks with respect to the real time $t = L_r(t)$ in this setting. We do not impose explicit restrictions on the progress speeds of the logical clocks in this chapter. However, we note that one can ensure smoothly progressing clocks by interpolation techniques, without weakening the synchronization guarantees.

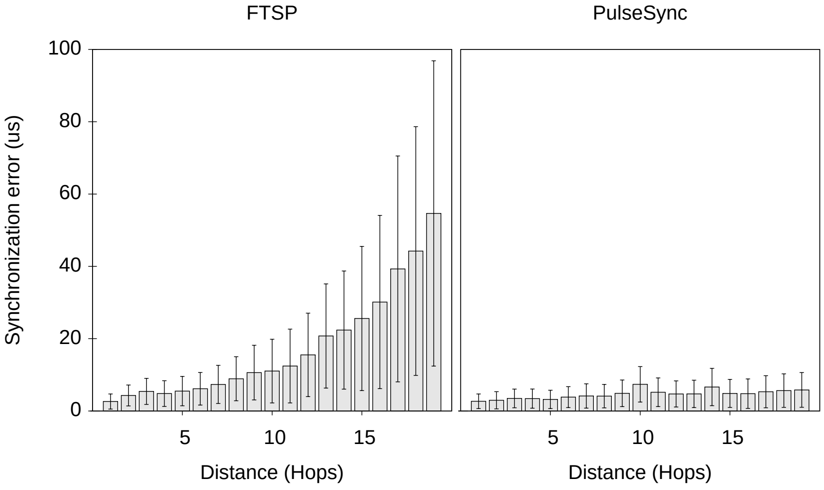

Figure 4.2: Synchronization error versus distance from the root node for FTSP (left) and PulseSync (right). See [63] for details on the testbed setup.

## 4.2 Overview

In the following, we study the probabilistic bounds that can be derived on the quality of global synchronization in the presented model. We begin by deriving a lower bound stating that on a path of length $d$ where on average $kd$, $k \in \mathbb{N}$, messages are transmitted in $kB$ time, the expected skew must be $\Omega(\mathcal{J}\sqrt{d/k})$. Essentially, this is a consequence of the fact that the variance of the transmission delays adds up linearly along the path to $\mathcal{J}^2 d$, whereas averaging over $k$ repeated transmissions reduces the variance by factor $k$. The resulting standard deviation thus must be in $\Omega(\mathcal{J}\sqrt{d/k})$. In our communication model, from this bound it can be derived that for any algorithm, the expected global skew must be $\Omega(\mathcal{J}\sqrt{D/k})$.

Opposed to that, we present *PulseSync*, an algorithm matching the stated lower bound. Basically, PulseSync floods estimates of the node $r$ through a breadth-first-search (BFS) tree rooted at $r$. All nodes strive to synchronize their clocks relative to $r$. In order to reduce the effect of the jitter, nodes keep track of the last $k$ received values and compute a regression line mapping the own hardware clock to the estimated clock values of the root node.

This approach is not new and has been implemented in the well known *Flooding Time Synchronization Protocol* (FTSP) [77]. However, the synchronization quality of FTSP becomes poor with growing network diameter due

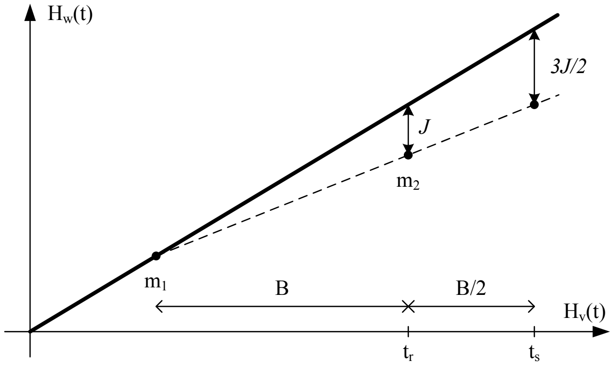

Figure 4.3: FTSP logical clock computation scheme for the special case of $k = 2$ data points. Here the first data point $m_1$ is perfectly accurate, while $m_2$ suffers an error of $\mathcal{J}$ because the respective message traveled slowly, i.e., the receiving node underestimated the time it took to transmit the message. Hence, the regression line has a value that is too small precisely by $\mathcal{J}$ at the receiving time $t_r$. FTSP nodes send clock updates in time slots that are chosen independently at random, once every $B$ time. Thus, at a time $t_s$, which is at least $t_r + B/2$ with probability 1/2, the node will send a message with clock estimate read from the regression line. This estimate will be $\mathcal{J} + (t_s - t_r)\mathcal{J}/B$ smaller than the true value, because the error of $\mathcal{J}$ on the value received at time $t_r$ also implies that the slope of the regression line is $\mathcal{J}/B$ too small. In summary, the error on the second received value is amplified by factor at least 3/2 with probability at least 1/2. Since sending slots are chosen independently, this happens independently with probability at least 1/2 at each hop, leading to an exponential amplification of errors.

to two reasons. Firstly, FTSP sends messages according to a random schedule, where nodes transmit one beacon every $B$ (local) time. Therefore, the expected time it takes until information propagates to a leaf in distance $D$ from the root is $DB/2$. In contrast, PulseSync aligns sending times in a pattern matching the BFS tree along which clock updates propagate, implying that—in practical networks—new clock values will reach all nodes within a single beacon interval $B$ (a "pulse"). Apart from reducing the global skew, this technique enables that logical clocks depend on a preceding time period of length $\Theta(kB)$ only, as opposed to $\Omega(DkB)$ for FTSP.[2]

[2]This is a result of forwarded clock values being read out of the regression constructed from the last $k$ received values. The dependency on very old values is weak, however,

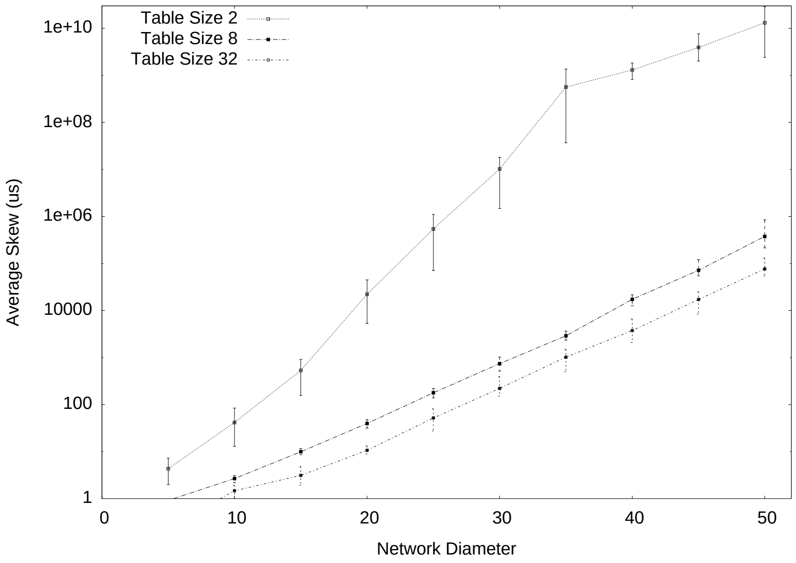

Figure 4.4: Simulation of FTSP for line topologies with different diameters using varying sizes $k \in \{2, 8, 32\}$ of the regression table. Errors clearly grow exponentially in the diameter, for all three values of $k$. Mean synchronization errors are averaged over five runs, error bars indicate values of runs with maximum and minimum outcome. See [63] for details.

Secondly, despite being designed as a multihop synchronisation protocol, FTSP exhibits exponentially growing global skew with respect to the network diameter (see Figure 4.2), rendering the protocol unsuitable for large-diameter deployments (which indeed occur in practice, cf. [46]). This undesired behavior is a result of the way the algorithm makes use of regression lines. The estimates nodes send to their children in the tree are read from the same regression line used to compute logical clocks. Thus, they are extrapolated from the previously received, erroneous estimates. This can lead to an amplification of errors exponential in the hop distance. For $k = 2$ this can easily be understood (see Figure 4.3). For larger $k$, the situation gets more complicated, but for any constant $k$ "bad luck" will frequently overcome the

$\Omega(D + k)$ of the most recent values contribute significantly to the outcome of the computation.

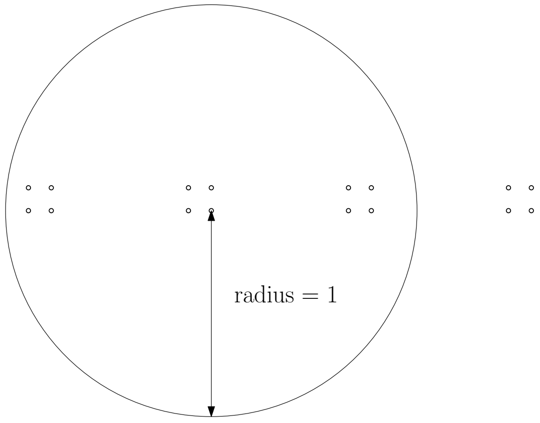

Figure 4.5: A simple unit disk graph (see Definition 9.7). Nodes are arranged into clusters of size four. The clusters form a line. Each node is connected to all nodes within its own and neighboring clusters.

dampening effect (see [96]) of the regression if $n$ is large. Moreover, simulation indicates that even for large values of $k$ the problem quickly becomes devastating when the network diameter grows (see Figure 4.4). This problem can be avoided if one uses independent estimates of the nodes' clock rates to compensate drift during the time period in which nodes locally increase received clock estimates until they can be forwarded to their children. Since PulseSync forwards clock values as quickly as possible, in our test setting it was already sufficient to rely on the unmodified hardware clock readings to resolve this issue. Nonetheless, we will prove that if nodes use independent clock estimates to compute approximations of their hardware clock rates, the bound on the global skew becomes entirely independent of hardware clock drift.

It has been argued that in some graphs one may exploit that the root is connected to each node by multiple paths. Consider for instance the unit disk graph in Figure 4.5. If the "clusters" transmit sequentially from left to right, each node could obtain multiple estimates of equal quality within a single pulse. This will decrease the variance of estimates by the number of nodes in each cluster. In general, one can express the possible gains of this strategy for any node $v \in V$ in terms of the resistance between the root $r$ and $v$ if each link in the network is replaced by a unit resistor [39]. However, since message size should remain small, this approach necessitates that nodes receive multiple messages in each beacon interval. This contradicts our goal of keeping energy consumption low. If on the other hand we accept a larger battery drain, we can achieve better synchronization by simply reducing $B$ and increasing $k$ accordingly (see Corollaries 4.8 and 4.9).

## 4.3 Lower Bound

In order to derive our lower bound on the global skew, we first examine how well a neighbor of the root can synchronize its clock with respect to $L_r$.

**Lemma 4.1.** *Assume that $r$ sends at most $k \in \mathbb{N}$ messages within a time interval $T := (t - kB, t] \subset \mathbb{R}_0^+$. Suppose $v \in \mathcal{N}_r$ computes an estimate $\hat{o}_v(t)$ of $o_v(t) := H_v(t) - t$ without—directly or indirectly—relying on any events preceding $T$. Then the probability that $\hat{o}_v(t)$ has an error of $\mathcal{J}/\sqrt{k}$ is at least constant, i.e.,*

$$P\left[|\hat{o}_v(t) - o_v(t)| \geq \frac{\mathcal{J}}{\sqrt{k}}\right] \in \Omega(1).$$

*Proof.* We claim that w.l.o.g. (*i*) no other nodes relay information about $r$'s clock values to $v$, (*ii*) $r$ sends all messages at time $t$ and (*iii*) each message contains only the clock value at the time of sending.

To see this, observe first that even if another node knew the state of $r$ exactly, it could not do better than $r$ itself as its messages are subject to the same variance in delay as $r$'s. Next, including several values into a single message does not help in estimating $o_v(t)$, as the crucial point is that the time of delivery of the message in comparison to the expected time of its delivery is unknown to both sender and receiver. Thus, all estimates that $v$ derives on $r$'s clock values are inflicted with exactly the same error due to jitter. Moreover, sending a different value than the one at the time of sending only meant that $v$ had to guess, based on its local clock and the messages from $r$, the value of $H_v(t')$ at the time $t'$ when $r$ read the respective clock value. This however could only reduce the quality of the estimate. As we deliberately lifted any restrictions $r$ had on sending times, there is no advantage in sending the message at a different time than $t$. Finally, since we excluded the use of any information on times earlier than $t - kB$ in the preconditions of the lemma, $r$ has no valuable information to share except its hardware clock reading at time $t$.

In summary, $r$ can at best send $k$ messages containing $t$ at time $t$, such that $v$ will learn that $r$ sent $k$ messages at time $t$ that have been registered at local times $H_v(t + X_i)$, where $X_i$, $i \in \{1, \ldots, k\}$, are independent normally distributed random variables with zero mean and variance $\mathcal{J}^2$. Since $X_i$ is unknown to $v$, it cannot determine $H_v(t) - t$. The best it can do is to read the $k$ values $H_v(t + X_i)$ and take each value $H_v(t + X_i) - t$ as an estimate. This can be interpreted as $k$ measurements of $o_v(t)$ suffering from independent normally distributed errors $h_v X_i \in \Theta(X_i)$ (as $|h_v - 1| = \rho_v \leq \rho$ and $\rho < 1$ is a constant). Hence, $\hat{o}_v(t)$ is (at best) the mean of $v$'s clock readings minus $t$. According to Lemma 2.19, this value is normally distributed with mean $o_v(t)$ and variance $\Theta(\mathcal{J}^2/k)$, which by Lemma 2.20 gives the claimed bound. □

At first glance, it seems tempting to circumvent this bound by just increasing the time interval information is taken from. Indeed this improves synchronization quality as long as the model assumption that clock rates and topology do not change remains (approximately) valid. As soon as conditions change quickly, the system will however require more time to adapt to the new situation, thus temporarily incurring larger clock skews.

The given bound on the synchronization quality between neighbors generalizes to multihop communication easily.

**Corollary 4.2.** *Given a shortest path $(v_0 := r, v_1, \ldots, v_d)$, assume that $kd$ messages, for some $k \in \mathbb{N}$, are sent and received by the nodes on the path within a time interval $T := (t - kB, t] \subset \mathbb{R}_0^+$. Suppose $v_d$ computes an estimate $\hat{o}_{v_d}(t)$ of its hardware clock offset $o_{v_d}(t)$ at time $t$ that does not rely on any events before $T$. Then the probability that $\hat{o}_{v_d}(t)$ has an error of $\mathcal{J}\sqrt{d/k}$ is constant, i.e.,*

$$P\left[|\hat{o}_{v_d}(t) - o_{v_d}(t)| \geq \frac{\mathcal{J}\sqrt{d}}{\sqrt{k}}\right] \in \Omega(1).$$

*Proof.* Assume w.l.o.g. that $h_{v_i} = 1$ for all $i \in \{1, \ldots, d\}$. Denote by $o_i := o_{v_i}(t) - o_{v_{i-1}}(t)$, $i \in \{1, \ldots, d\}$ the offset between the clocks of $v_i$ and $v_{i-1}$. Consider the following scheme. First $v_1$ determines an estimate $\hat{o}_1(t)$ of $o_{v_1}(t) = o_1$, then $v_2$ an estimate $\hat{o}_2$ of the offset $o_2$ towards $v_1$, and so on. Thus, by incorporating the results into the messages, $v_i$, $i \in \{1, \ldots, d\}$, can estimate $o_{v_i}(t)$ by $\hat{o}_{v_i}(t) = \sum_{j=1}^{i} \hat{o}_j(t)$. Since clocks do not drift and there are no "shortcuts" as $(v_0, \ldots, v_d)$ is a shortest path, this scheme is at least as good as an optimal one (obeying the model constraints). Let $k_i$, $i \in \{1, \ldots, d\}$, denote the number of messages node $v_i$ receives from its predecessor. As seen in the proof of Lemma 4.1, $\hat{o}_i$ is normally distributed with mean $o_i$ and variance $\mathcal{J}^2/k_i$. By Lemma 2.19, it follows that $\hat{o}_{v_d}$ is normally distributed with mean $o_{v_d}$ and variance $\sum_{i=1}^{d} \mathcal{J}^2/k_i$. Because $\sum_{i=1}^{d} k_i = kd$, this variance is minimized by the choice $k_i = k$ for all $i \in \{1, \ldots, d\}$. We get that

$$\operatorname{Var}[\hat{o}_{v_d}(t)] \geq \mathcal{J}^2 d/k,$$

which by Lemma 2.20 yields the desired statement. □

Next, we infer our lower bound on the global skew.

**Theorem 4.3.** *Suppose that $k \in \mathbb{N}$ and each node sends and receives on average at most one message in $B$ time. If a clock synchronization algorithm determines $L_v(t)$ at all nodes $v \in V$ and times $t \in \mathbb{R}_0^+$ depending on events that happened after time $t - kB$ only, then at uniformly random times $t$ from a sufficiently large time interval we have that*

$$\mathbb{E}[|L_v(t) - t|] \in \Omega\left(\frac{\mathcal{J}\sqrt{d}}{\sqrt{k}}\right),$$

*where $d$ is the distance of $v$ from the root.*

*Proof.* Let $(v_0 := r, v_1, \ldots, v_d := v)$ denote a shortest path from $r$ to $v$. Because all nodes receive on average at most one message in $B$ time, for symmetry reasons we may w.l.o.g. assume that all estimates $v$ obtains on its offset depend on messages along this path only. Let $\mathcal{E}$ be the event that at a time $t$ sampled uniformly at random from a sufficiently large time period it holds that the nodes $v_i$, $i \in \{0, \ldots, d-1\}$, sent and received in total at most $2kd$ messages during the interval $(t - kB, t]$. Because nodes send and receive on average at most one message in $B$ time, linearity of expectation and Markov's bound imply that the probability of $\mathcal{E}$ must be at least $1/2$. By Corollary 4.2, we have that any estimate $v$ may compute of $H_v(t) - t$ has an error of $\mathcal{J}\sqrt{d}/\sqrt{2k}$ with at least constant probability, proving the claim. □

Seen from a different angle, this result states how quickly the system may adapt to dynamics. It demonstrates a trade-off between the contradicting goals of minimizing message frequency, global skew, and the time period logical clock values depend on. Given a certain stability of clock rates and having fixed a desired bound on the global skew, for instance, one can derive a lower bound on the number of messages nodes must at least send in a given time period to meet these conditions. Similarly, the theorem yields a lower bound on the time span it takes until a node (re)joining the network may achieve optimal synchronization for a given message frequency, granted that the other nodes make no additional effort to support this end.

## 4.4 PulseSync

The central idea of the algorithm is to distribute information on clock values as fast as possible, while minimizing the number of messages required to do so. In particular, we would like to avoid that it takes $\Omega(BD)$ time until distant nodes learn about clock values broadcast by the root node $r$. Obviously, a node cannot forward any information it has not received yet, enforcing that information flow is directed. An intermediate node on a line topology has to wait for at least one message from a neighbor. On the other hand, after reception of a message it ought to forward the derived estimate as quickly as possible in order to spread the new knowledge throughout the network. Thus, we naturally end up with flooding a *pulse* through the network. In order to keep the number of hops small, the flooding takes place on a breadth-first search tree.

To keep clock skews small at all times, each node $v \in V$ does not only minimize its offset towards the root whenever receiving a message, but also employs a drift compensation, i.e., tries to estimate $h_v$ and increase its logical clock at the speed of $H_v$ divided by this estimate. Considering that we modeled $H_v$ as an affine linear function and the fluctuations of message delays as independently normally distributed random variables, linear regression is a canonical choice as a means to compute $L_v(t)$ out of $H_v(t)$ and the last $k$ clock updates received.

We need to specify how nodes that are not children of the root obtain accurate estimates of $r$'s clock. Recall that nodes are not able to send a message at arbitrary times. Thus, it is necessary to account for the time span $\tau_v$ that passes between the time when node $v \in V$ receives a clock estimate from a parent and the time when it can send a (derived) estimate to its children. The most simple approach here is that if $v$ obtains an estimate $\hat{t}$ of the root's clock value $L_r(t) = t$ from a message received at time $t$, it sends at time $t + \tau_v$ the value

$$\hat{t} + (H_v(t + \tau_v) - H_v(t))$$

to its children. Thus, the quality of the estimate will deteriorate by at most

$$|(H_v(t + \tau_v) - H_v(t)) - ((t + \tau_v) - t)| = |h_v - 1|\tau_v \leq \rho\tau_v.$$

We will refer to this as *simple forwarding.* Intuitively, granted that $\tau_v$ is small enough, i.e., $\max_{v \in V}\{\rho_v\tau_v\} \ll \mathcal{J}/\sqrt{D}$ (here $\sqrt{D}$ comes into play as jitters are likely to cancel out partially), the additional error introduced by simple forwarding is dominated by message jitter and thus negligible.

In our test setting, this technique already turned out to be sufficient for achieving good results. However, this might not be true in general, due to different hardware, larger networks, harsh environments, etc. Hence we devise a slightly more sophisticated scheme we call *stabilized forwarding.* As discussed before, it is fatal to replace the term $H_v(t + \tau_v) - H_v(t)$ by $L_v(t + \tau_v) - L_v(t)$, i.e., approximate the progress of real time by means of the regression line that is computed partially based on the estimate $\hat{t}$ obtained at time $t$. Instead, we use an independent estimate $\hat{h}_v$ of $h_v$ to compensate the drift. To this end, given $k \in 2\mathbb{N}$, node $v \in V$ computes the regression line defining $L_v$ according to the $k/2$ most recent messages only. The remaining $k/2$ messages nodes may take information from are used to provide clock estimates with simple forwarding. From these values a second regression line is determined, whose slope $s$ should be close to $1/h_v$. As we know that $h_v \in [1-\rho, 1+\rho]$, nodes set $\hat{h}_v$ to $1-\rho$ if the outcome is too small and to $1+\rho$ if it is too big. All in all,

$$\hat{h}_v := \begin{cases} 1-\rho & \text{if } 1/s \leq 1-\rho \\ 1+\rho & \text{if } 1/s \geq 1+\rho \\ 1/s & \text{otherwise.} \end{cases}$$

Apart from sending $\hat{t} + H_v(t+\tau_v) - H_v(t)$ at time $t+\tau_v$ after receiving a message at time $t$, node $v$ now also includes the value

$$\hat{t} + \frac{H_v(t+\tau_v) - H_v(t)}{\hat{h}_v}$$

into the message. This (usually) more precise estimate is then used to derive the regression line defining $L_v$ from the $k/2$ most recent messages. Obviously, one cannot use stabilized forwarding until nodes received sufficiently many clock estimates; for simplicity, we disregard this in the pseudocode of the algorithm. We remark that a similar approach has been proposed for high latency networks where the drift during message transfer is a major source of error [101].

The pseudocode of the algorithm for non-root nodes is given in Algorithm 4.2, whereas the root follows Algorithm 4.1. In the abstract setting, a message needs to contain the two estimates of the root's clock value only. For clarity, we utilize sequence numbers $i \in \mathbb{N}$, initialized to one, in the pseudocode of the algorithm. In practice, a message may contain additional useful information, such as an identifier, the identifier of the (current) root, or the (current) depth of a node in the tree. For the root node, the logical clock is simply identical to the hardware clock. Any other node computes $L_v(t)$ as the linear regression of the $k/2$ most recently stored pairs of hardware clock values and the corresponding estimates with stabilized forwarding, evaluated at $H_v(t)$. As stated before, the value $\hat{h}_v(t)$ is computed out of the $k/2$ estimates with simple forwarding from the preceding pulses, as the inverse slope of the linear regression of these values.

**Algorithm 4.1**: Whenever $H_r(t) \bmod B = 0$ at the root node $r$.

```
1 wait until time $t+\tau_r$ when allowed to send
2 send $\langle t+\tau_r, t+\tau_r, i\rangle$ // recall that $H_r(t+\tau_r) = t+\tau_r$
3 $i := i+1$
```

**Algorithm 4.2**: Node $v \neq r$ receives its parent's message $\langle \hat{t}, \tilde{t}, i \rangle$ with sequence number $i$ at local time $H_v(t)$.

```
1 delete $\langle \cdot, \cdot, \cdot, i-k+1 \rangle$
2 store $\langle H_v(t), \hat{t}, \tilde{t}, i \rangle$
3 wait until time $t + \tau_v$ when allowed to send
4 send $\langle \hat{t} + H_v(t+\tau_v) - H_v(t), \tilde{t} + (H_v(t+\tau_v) - H_v(t))/\hat{h}_v, i \rangle$
5 $i := i + 1$
```

## 4.5 Analysis

In this section, we will prove a strong probabilistic upper bound on the global skew of PulseSync. To this end, we will first derive a bound on the accuracy of the estimates nodes compute of their hardware clock rates. Then we will proceed to bounding the clock skews themselves.

**Definition 4.4** (Pulses). *Pulse $i \in \mathbb{N}$ is* complete *when all messages with sequence number $i$ have been sent. We say that pulses are* locally separated *if for all $i \in \mathbb{N}$ each node sends its message with sequence number $i$ at least $\alpha B$ time before receiving the one with sequence number $i+1$, where $\alpha \in \mathbb{R}^+$ is a constant.*

After the initialization phase is over, i.e., as soon as all nodes could fill their regression tables, nodes are likely to have good estimates on their clock rates. Interestingly, the quality of the estimates is independent of the hardware clock drifts, as the respective systematic errors are the same for all estimates of the root's clock and thus cancel out when computing the slope of the regression line.

**Lemma 4.5.** *For $v \in V$ and arbitrary $\delta \in \mathbb{R}^+$ define*

$$\Delta_h := \min \left\{ 2\rho, \frac{\delta \mathcal{J} \sqrt{D}}{k^{3/2} B} \right\}.$$

*Suppose that pulses are locally separated. Then, at any time $t$ when at least $k \in 2\mathbb{N}$ pulses are complete, it holds that*

$$P\left[ \left| \frac{h_v}{\hat{h}_v(t)} - 1 \right| \leq \Delta_h \right] \in 1 - e^{-\Omega(\delta^2 \log \delta)}.$$

*Proof.* Assume that $(v_0 := r, v_1, \ldots, v_d := v)$ is the path from $r$ to $v$ in the BFS tree (i.e., in particular $d \leq D$). Consider a simply forwarded estimate $\hat{t}$ that has been received by $v$ at time $t$. Backtracking the sequence of messages leading to this value and applying Lemma 2.19, we see that $r$ sent its respective message at a time that is normally distributed around $t-\sum_{i=0}^{d-1}\tau_{v_i}$ with variance $d\mathcal{J}^2$. Thus, since node $v_i$ increases each simply forwarded estimate at rate $h_{v_i}$, $\hat{t}-t$ is normally distributed with mean $\sum_{i=0}^{d-1}(h_{v_i}-1)\tau_{v_i}$ and variance at most

$$\sum_{i=0}^{d-1}((1+\rho_{v_i})\mathcal{J})^2 \leq 4d\mathcal{J}^2.$$

By Theorem 2.22, thus the slope $\hat{s}$ of the regression line $v$ computes is normally distributed with mean $1/h_v$ and variance

$$\mathcal{O}\left(\frac{d\mathcal{J}^2}{h_v k^3 B^2}\right) \subseteq \mathcal{O}\left(\frac{D\mathcal{J}^2}{k^3 B^2}\right).$$

Here we used that pulses are locally separated, implying that $\sum_{i=1}^{N}(x_i-\bar{x})^2 \in \Omega(h_v k^3 B^2)$ (in terms of the notation from the theorem). Recall that $h_v \geq 1-\rho_v \geq 1-\rho > 0$ and we made sure that $\hat{h}_v \in [1-\rho, 1+\rho]$. Thus, we can infer that the error $|h_v/\hat{h}_v - 1| = |h_v s - 1|$ is bounded both by $1/(1-\rho) \in \mathcal{O}(1)$ times the deviation of the slope from its mean and $2\rho$. Hence, the claim follows by Lemma 2.20. □

Based on the preceding observations, we can now prove a bound on the skew between a node and the root.

**Theorem 4.6.** *Suppose pulses are locally separated. Denote by* $(v_0 := r, v_1, \ldots, v_d := v)$ *the shortest path from the root to* $v \in V$ *along which estimates of* $r$*'s clock values are forwarded. Set* $\mathcal{T}_v := \sum_{i=1}^{d}\tau_{v_i}$*, i.e., the expected time an estimate "travels" along the path. Suppose* $t_1 < t_2$ *are two consecutive times when* $v$ *receives a message and suppose that at time* $t_1$ *at least* $3k/2$*,* $k \in 2\mathbb{N}$*, pulses are complete. Then for any* $\delta, \varepsilon \in \mathbb{R}^+$ *and* $\Delta_h$ *as in Lemma 4.5 it holds that*

$$P\left[\forall t \in [t_1, t_2) : |L_v(t) - t| \leq \varepsilon\mathcal{J}\sqrt{\tfrac{D}{k}} + \Delta_h \mathcal{T}_v\right]$$
$$\in 1 - \tfrac{kD}{2}e^{-\Omega(\delta^2\log\delta)} - e^{-\Omega(\varepsilon^2\log\varepsilon)}.$$

*Proof.* Since at least $3k/2$ pulses are complete, according to Lemma 4.5 during the last $k/2$ pulses we had at any time $t$ and for any node $v_i$, $i \in \{0, \ldots, d-1\}$ that $|h_{v_i}/\hat{h}_{v_i}(t) - 1| \leq \Delta_h$ with probability $1 - e^{-\Omega(\delta^2\log\delta)}$. Denote by $\mathcal{E}$ the event that the last $k/2$ estimates with stabilized forwarding that $v$ received have been increased at *all* nodes on the way at a rate differing by no more than $\Delta_h$ from 1. Since $\hat{h}_{v_i}$ only changes when a message is received, we can apply the union bound to see that $\mathcal{E}$ occurs with probability at least $1 - kDe^{-\Omega(\delta^2\log\delta)}/2$.

Assume now that $\mathcal{E}$ happened and also that $1 - kDe^{-\Omega(\delta^2 \log \delta)}/2 > 0$ (as otherwise the bound is trivially satisfied). Consider the errors of the above $k/2$ estimates. Each estimate has been "on the way" for expected time $\mathcal{T}_v$, i.e., the absolute of the mean of its error is bounded by $\Delta_h \mathcal{T}_v$. The remaining fraction of the error is due to message jitter. Note that if the estimates $\hat{h}_{v_i}$ are very bad, this might amplify the effect of message jitter. However, since the $\hat{h}_{v_i}$ are uniformly bounded by $1-\rho$ and $1+\rho$, we can account for this effect by multiplying $\mathcal{J}$ with a constant. Thus, we can still assume that at each hop, a normally distributed random variable with zero mean and variance $\mathcal{O}(\mathcal{J}^2)$ is added to the respective estimate of $r$'s current clock value, yielding a random experiment which stochastically dominates the true setting (with respect to clock skews). In summary, by Lemma 2.19 w.l.o.g. each estimate that $v$ obtains suffers an independently and normally distributed error with mean $\mu \in [-\Delta_h \mathcal{T}_v, \Delta_h \mathcal{T}_v]$ and variance $\mathcal{O}(d\mathcal{J}^2) \subseteq \mathcal{O}(D\mathcal{J}^2)$.

By Theorem 2.22, the slope of the regression line utilized to compute $L_v$ suffers a normally distributed error of zero mean and standard deviation $\mathcal{O}(\mathcal{J}\sqrt{D}/(k^{3/2}B))$. Denote by $\bar{t}$ the mean of the times when $v$ received the $k/2$ messages it computes the regression from. As for all times $\bar{t} < t \in [t_1, t_2)$ we have that $t - \bar{t} \leq (k/2+1)B$, we can bound[3]

$$|L_v(t) - t| \leq |L_v(\bar{t}) - \bar{t}| + \left| \left( \frac{h_v(t)}{\hat{h}_v} - 1 \right) (\bar{t} - t) \right|,$$

where the second term is normally distributed with zero mean and standard deviation $\mathcal{O}(\mathcal{J}\sqrt{D/k})$.

Again by Theorem 2.22, the deviation of the line at the time $\bar{t}$ itself is normally distributed with mean $\mu$ and a standard deviation of $\mathcal{O}(\mathcal{J}\sqrt{D/k})$. Thus, applying Lemma 2.20 and the union bound yields that, conditional to $\mathcal{E}$, the event $\mathcal{E}'$ that

$$\forall t \in [t_1, t_2) : \ |L_v(t) - t| \leq \varepsilon \mathcal{J} \sqrt{\frac{D}{k}} + \Delta_h \mathcal{T}_v$$

occurs with probability at least $1 - e^{-\Omega(\varepsilon^2 \log \varepsilon)}$. We conclude that

$$\begin{array}{rcl} P[\mathcal{E}'] & \geq & P[\mathcal{E}] \cdot P[\mathcal{E}' | \mathcal{E}] \\ & \in & \left(1 - \frac{kD}{2} e^{-\Omega(\delta^2 \log \delta)}\right) \left(1 - e^{-\Omega(\varepsilon^2 \log \varepsilon)}\right) \\ & \subseteq & 1 - \frac{kD}{2} e^{-\Omega(\delta^2 \log \delta)} - e^{-\Omega(\varepsilon^2 \log \varepsilon)} \end{array}$$

as claimed. □

[3]Note that the use of the expression $L_v(\bar{t})$ here is an abuse of notation, as we refer to the $y$-value the regression line that $v$ computes at time $t$ assigns to $x$-value $H_v(\bar{t})$.

The term $\mathcal{T}_v$ occurring in the bound provided by the theorem motivates the following definition.

**Definition 4.7** (Pulse Time)**.** *Denote for $v \in V$ by $(v_0 := r, v_1, \ldots, v_d := v)$ the shortest path from $r$ to $v$ along which PulseSync sends messages and set $\mathcal{T}_v := \sum_{i=0}^{d-1} \tau_{v_i}$. The* pulse time *then is defined as*

$$\mathcal{T} := \max_{v \in V} \{\mathcal{T}_v\}.$$

Theorem 4.6 implies that the proposed technique is indeed optimal provided a comparatively weak relation between $B$ and $P$ is satisfied.

**Corollary 4.8.** *Suppose pulses are locally separated and that*

$$B \geq \frac{\mathcal{T}}{k}\sqrt{\frac{\log(kD)}{\log\log(kD)}}.$$

*Then the expected clock skew of any node $v \in V$ in distance $d$ from the root at any time $t$ when at least $3k/2$ pulses are complete is bounded by*

$$\mathbb{E}[|L_v(t) - t|] \in \mathcal{O}\left(\mathcal{J}\sqrt{\frac{d}{k}}\right).$$

*Proof.* W.l.o.g., we assume that $d = D$ (otherwise just consider the subgraph induced by all nodes within distance $d$ from $r$). For $i \in \mathbb{N}$, set

$$\Delta_h(i) := \frac{i\mathcal{J}}{k^{3/2}B}\sqrt{\frac{\log(kD)D}{\log\log(kD)}}.$$

By assumption, we have that

$$\Delta_h(i)\mathcal{T}_v \leq \frac{i\mathcal{J}\mathcal{T}}{k^{3/2}B}\sqrt{\frac{\log(kD)D}{\log\log(kD)}} \leq i\mathcal{J}\sqrt{\frac{D}{k}},$$

giving by Theorem 4.6 for $\delta = i\sqrt{\log(kD)/\log\log(kD)}$ and $\varepsilon = i$ that

$$P\left[|L_v(t) - t| > 2i\mathcal{J}\sqrt{\frac{D}{k}}\right] \in e^{-\Omega(i^2 \log i)}.$$

It follows that

$$\begin{aligned}
\mathbb{E}[|L_v(t) - t|] &\leq \sum_{i=0}^{\infty} P\left[|L_v(t) - t| > 2i\mathcal{J}\sqrt{\frac{D}{k}}\right] 2\mathcal{J}\sqrt{\frac{D}{k}} \\
&\in \left(1 + \sum_{i=1}^{\infty} e^{-\Omega(i^2 \log i)}\right) 2\mathcal{J}\sqrt{\frac{D}{k}} \\
&\subseteq \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D}{k}}\right).
\end{aligned}$$

□

Note that Corollary 4.8 requires a *lower* bound on $B$, while in practice we are interested in choosing $B$ large to minimize energy consumption. Of course, a beacon interval that is too large is undesired because one wants the system to adapt quickly to dynamics. However, the pulse time is a trivial lower bound on this response time, i.e., it does not make sense to choose $Bk \in o(\mathcal{T})$. Thus, we remain with a small gap of $\mathcal{O}\left(\sqrt{\log(kD)/\log\log(kD)}\right)$ to countervail the fact that the drift compensations the nodes on a path of length $D$ employ are dependent, making best use of the limited number of recent clock estimates that are available.

In practice, it is important to bound clock skews at *all* times and *all* nodes, as algorithms may fail if presumed bounds are violated even once. Naturally, a probabilistic bound cannot hold with certainty; indeed, in our model arbitrary large skews must occur if we just wait for sufficiently long. However, for time intervals that are bounded by a polynomial in $n$ times $B$ we can state a strong bound that holds w.h.p.

**Corollary 4.9.** *For $i \in \mathbb{N}$, let $t_i$ denote the time when the $i^{th}$ pulse is complete. Provided that the prerequisites of Theorem 4.6 are satisfied, the total number of pulses $p$ is polynomial in $n$, and $B \geq \mathcal{T}/k$, we have that*

$$\max_{t \in [t_{3k/2},\, t_p]} \{\mathcal{G}(t)\} \in \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D\log n}{k \log\log n}}\right)$$

*w.h.p.*

*Proof.* Observe that $3k/2 \leq p$ or nothing is to show as $t_p < t_{3k/2}$. As also $D < n$, we have that $kD$ is polynomially bounded in $n$. Thus, values $\delta, \varepsilon \in \mathcal{O}\left(\sqrt{\log n/\log\log n}\right)$ exist such that

$$1 - \frac{kD}{2}e^{-\Omega(\delta^2 \log \delta)} - e^{-\Omega(\varepsilon^2 \log \varepsilon)} \geq 1 - \frac{1}{pn^{c+1}}.$$

We apply Theorem 4.6 to each node $v \in V$ and each pulse $i \in [3k/2, p]$. Due to the bound $B \geq \mathcal{T}/k \geq \mathcal{T}_v/k$ and the definition of $\Delta_h$, we get for all times $t$ from the respective pulse that

$$|L_v(t) - t| \in \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D\log n}{k\log\log n}} + \Delta_h \mathcal{T}_v\right) \subseteq \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D\log n}{k\log\log n}}\right)$$

with probability at least $1 - 1/(pn^{c+1})$. The statement of the corollary then follows by the union bound applied to all nodes and all pulses $i \in [3k/2, p]$. □

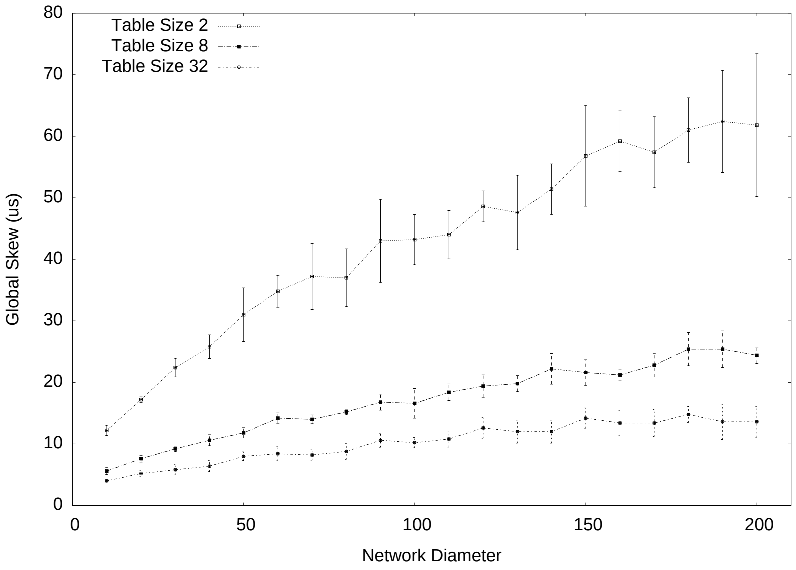

Figure 4.6: Simulations of PulseSync on line topologies. Results agree well with predictions. In particular, it can be observed that the global skew grows roughly as the square-root of the network diameter. For details the reader is referred to [63].

Note that when considering all nodes, the lower bound on $B$ relaxes to $\mathcal{T}/k$, i.e., we can achieve the stated bound despite the fastest possible adaption to dynamics. Moreover, comparing the previous bounds to the results from simulation and implementation of the algorithm (Figures 4.6 and 4.7), we find the predictions on the synchronization quality of the algorithm well met.

## 4.6 Concluding Remarks

In this chapter, we proposed a clock synchronization algorithm that exhibits asymptotically optimal synchronization quality in the given model. Testbed results from a prototypical implementation indicate that our abstract model is appropriate for capturing crucial aspects of sensor networks. Considering that we derive skew bounds that are independent of hardware clock drifts

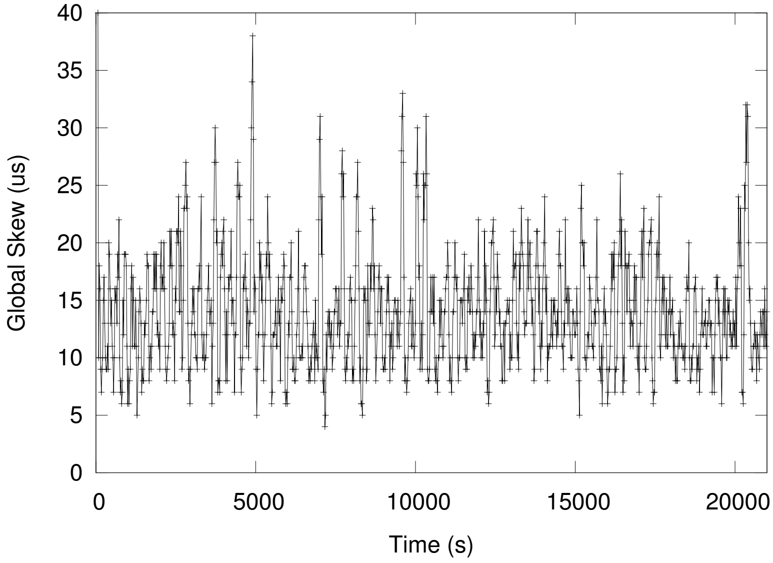

Figure 4.7: Global skew of an execution of PulseSync with $B = 30$ s and $k = 8$ on a line topology comprising 20 Mica2 sensor nodes. For times $t > 100$ s, we observed a maximum skew of 38 $\mu$s. See [63] for details.

provided that jitter is not too large, our results may also be of interest for high-latency networks, e.g. acoustic underwater networks.

However, our presentation has not been exhaustive in the sense that one can effortlessly derive a protocol suitable for practice, as several issues still need to be resolved. The test runs of the algorithm were executed on a simple line topology in a controlled environment. In order to finish pulses quickly on arbitrary topologies, an efficient broadcast routine is in demand. This problem has been studied extensively and is by now well-understood [7, 17, 18, 21, 87], hence we refrain from a discussion here. We confine ourselves to mentioning that it is sufficient to solve the broadcast problem on a sparse backbone of the network, since the remaining nodes may simply listen to pulses and derive their clocks without ever transmitting by themselves. Computing such a backbone, i.e., a connected dominating set, can also be solved efficiently [98].

Finally, in order to exploit the full potential of the algorithm, a good implementation must deal with message loss, changes in topology, and varying hardware clock rates. Ideally, one would choose $kB$ adaptively, reducing it in face of high volatility of clock rates and/or connectivity and increasing $k$ (to gain synchronization quality) or $B$ (to save energy) again when the system becomes more stable.

# Chapter 5

# Gradient Clock Synchronization

*"I got used to switching off once they started talking about this."*
*– Yvonne-Anne Pignolet on my lively discussions with Thomas Locher regarding clock synchronization.*

In the previous chapter, we considered clock synchronization in the context of a concrete system. Exploiting the properties of wireless networks, we obtained a bound of $\mathcal{O}(\sqrt{D \log n / \log\log n})$ on the global skew that holds w.h.p. This result is however fragile with respect to changes in the model. We made strong assumptions, in particular independently normally distributed deviations in message transmission times, constant clock drifts, and fixed topology. Moreover, PulseSync does not attempt to minimize the local skew. In fact, in a circle the two leafs of a BFS tree are likely to be the pair of nodes experiencing the largest clock skews, yet they are neighbors.

We drop the former assumptions in favor of a worst-case model with regard to communication, clock drifts, and topology changes. We will devise an algorithm featuring an optimal *gradient property* as introduced in [33], i.e., the worst-case skew between any two nodes that have been connected by a path of length $d$ for sufficiently long is an asymptotically minimal function of $d$. At the same time, the algorithm is capable of extending the gradient property to newly appearing edges as quickly as possible.

This chapter is based on joint work with Fabian Kuhn, Thomas Locher, and Rotem Oshman [50]. In large parts, it builds on a preceding line of publications together with Thomas Locher [58, 59, 60]. The proposed gradient clock synchronization algorithm and its analysis have been extended from the model considered in these articles to the one introduced by Kuhn et al. [52], which differs mainly in that the graph changes dynamically. A proof of the gradient property of the algorithm in the general model is given in [51]; we strive for a simplified, more accessible presentation of the core concepts in this thesis. For a detailed introduction to the static setting and its analysis we refer the interested reader to [71].

## 5.1 Model

We point out that in the sequel we will adopt a model as simple as possible to still capture the main aspects of the problem. As a consequence, some of the assumptions in the model may seem artificial at first glance. For the sake of a straightforward presentation, we postpone the respective discussion to Section 5.6.

Similarly to Chapter 4, each device $v \in V$ is equipped with a differentiable hardware clock $H_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, where $H_v(0) := 0$. The drift of $H_v$ is bounded, i.e.,

$$h_v(t) := \frac{d}{dt}H_v(t) \in [1-\rho, 1+\rho],$$

where $0 < \rho \ll 1$ is the *maximum clock drift*, or simply *drift*. As before, $v$ does neither know the real time $t$ nor the rate at which $H_v$ (currently) increases, but may read $H_v(t)$. At all times $t$, node $v$ must compute a (differentiable) logical clock $L_v(t)$, where $L_v(0) := 0$. We require that the logical clock $L_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ also progresses at a controlled speed, i.e., there is a $\mu \in \mathcal{O}(1)$ such that

$$l_v(t) := \frac{d}{dt}L_v(t) \in [1-\rho, (1+\mu)(1+\rho)]. \tag{5.1}$$

Node $v$ is called *fast at time $t$* if $l_v(t) = (1+\mu)h_v(t)$ and *slow at time $t$* if $l_v(t) = h_v(t)$.

The margin $\mu$ is introduced to permit $v$ to increase its logical clock by a factor of $1+\mu$ faster than its hardware clock in order to catch up with other nodes. Since nodes do not know $h_v(t)$, the condition on $l_v$ imposes that $l_v(t) \in [h_v(t), (1+\mu)h_v(t)]$ at all times. Thus it must hold that

$$\mu > \frac{2\rho}{1-\rho},$$

as otherwise it might happen that

$$(1+\mu)h_v(t) \leq \left(1 + \frac{2\rho}{1-\rho}\right)(1-\rho) = 1+\rho = h_w(t),$$

for some $v, w \in V$, rendering it impossible for $v$ to catch up with $w$.

We have not specified yet what information $v \in V$ may access in order to compute $L_v(t)$. Apart from its hardware clock readings, each node $v \in V$ has a dynamic set of neighbors $\mathcal{N}_v : \mathbb{R}_0^+ \to \binom{V}{2}$. This relationship is symmetric, i.e., it induces a simple dynamic graph $G(t) = (V, E(t))$, $t \in \mathbb{R}_0^+$. We say that edge $e$ *exists during the time interval* $[t_1, t_2]$ if $e \in E(t)$ for all $t \in [t_1, t_2]$. The statement that $\{v, w\} \in E(t)$ at time $t$ is equivalent to $v$ having an estimate $\hat{L}_v^w(t)$ of $w$'s logical clock value $L_w(t)$ at time $t$ and vice versa. This estimate is inaccurate; it may be off by the *uncertainty* $U \in \mathbb{R}^+$, i.e.,

$$\forall t \in \mathbb{R}_0^+ : \ |\hat{L}_v^w(t) - L_w(t)| \leq U.$$

The value of $U$ depends on several factors, such as fluctuations in message transmission times, the manner in which estimates are obtained, clock drifts, the frequency at which nodes communicate in order to update their estimates, etc.

A fundamental lower bound [16] shows that in a static network, the global skew grows linearly with the network diameter. In dynamic networks there is no immediate equivalent to a diameter. Informally, the diameter corresponds to the number of hops it takes (at most) for information to spread from one end of the network to the other. To formalize this idea we adopt the following definitions.

**Definition 5.1** (Flooding). *A* flooding *that originates at node $v$ is a process initiated when node $v$ sends a* flooding message *to all its neighbors. We normalize the maximal message delay, i.e., each individual message is in transit for at most one time unit. Each node that receives the message for the first time forwards it immediately to all its neighbors. We say that the flooding is* complete *when all nodes have received a flooding message.*

**Definition 5.2** (Diameter and Flooding Jitter). *We say that the dynamic graph $G$ has a* diameter *of $D$ if a flooding originating at any node and any time always completes in at most $D$ time units. It has a* flooding jitter *of $\mathcal{J}_D \in [0, D]$ if each node can determine up to $\mathcal{J}_D$ the time span between a flooding being initiated and the node receiving the first flooding message.*

Note that in general $\mathcal{J}_D$ might be very different from $DU$. On the one hand, nodes might e.g. use reference broadcasts [30, 55] to obtain local, accurate estimates $U$, while flooding suffers from large jitters $\mathcal{J}$ at each hop, implying that $\mathcal{J}_D \geq D\mathcal{J} \gg DU$.[1] On the other hand, nodes might derive local estimates from infrequent direct communication every $\tau$ time. Observe that $U \geq 2\rho\tau$ because logical clocks must run at least at the hardware clock speed, i.e., in absence of new information estimated and true clock values may drift apart at rate $2\rho$. Thus, if $\tau \gg 1 + \mathcal{J}/\rho$, we have $DU \geq 2\rho\tau D \gg (\rho + \mathcal{J})D$. As it is possible to ensure that $\mathcal{J}_D \in \mathcal{O}((\rho + \mathcal{J})D)$, in this case we have $\mathcal{J}_D \ll DU$.

[1]The first inequality follows from the fact that in the worst case either all jitters may increase the time the flooding takes or all may decrease it. This is formalized by well-known shifting arguments (cf. [74]).

To measure the quality of an algorithm, we consider two kinds of requirements: a *global skew constraint* which gives a bound on the difference between any two logical clock values in the system, and a *dynamic gradient skew constraint* which becomes stronger if nodes have been connected by a short path for sufficiently long time. In particular, for nodes that remain neighbors for a long time, the dynamic gradient skew constraint requires a much smaller skew than the global skew constraint.

In the probabilistic model of the previous chapter, we considered the maximum clock skew $\mathcal{G}(t)$ at a given time $t$, since skews could become arbitrarily large. In the worst-case setting it makes sense to use the following more stringent definition.

**Definition 5.3** (Global Skew). *A clock synchronization algorithm $\mathcal{A}$ has a* global skew *of $\mathcal{G}(D, \mathcal{J}_D, \rho)$, if for any execution of $\mathcal{A}$ with drift $\rho$ and flooding jitter $\mathcal{J}_D$ on any dynamic graph of diameter $D$ it holds that*

$$\forall v, w \in V, t \in \mathbb{R}_0^+ : \ |L_v(t) - L_w(t)| \leq \mathcal{G}.$$

*For notational convenience, we omit the arguments $D$, $\mathcal{J}_D$, and $\rho$ of $\mathcal{G}$ in the following.*

Informally, no matter what the graph or the execution, skews should at all times be bounded by a function of the parameters $D$, $\mathcal{J}_D$, and $\rho$ only.

An ideal dynamic gradient skew constraint would state that any two nodes that are close in $G(t)$ have closely synchronized logical clocks. Apparently, this is not possible immediately after a "shortcut" has been formed, as it needs time to reduce skews between previously distant nodes. Rather, any path between two nodes $v, w \in V$ that has been part of the graph for sufficiently long imposes a constraint on $|L_v - L_w|$ that is tighter the shorter the path is. To state precisely what "sufficiently long" means, we need the following definition.

**Definition 5.4** (Stable Subgraphs). *For $T \in \mathbb{R}^+$, the $T$-*stable subgraph *of $G$ is defined as*

$$G_T(t) := \left( V, \left\{ e \in \binom{V}{2} \,\middle|\, \forall t' \in [t-T, t] : e \in E(t') \right\} \right).$$

*For two nodes $v, w \in V$ we denote by $d_T(v, w)$ the distance of $v$ and $w$ in $G_T$.*

We could now formulate the dynamic gradient skew as a function bounding the skew on each path in terms of its length and the time span it existed without interruption. However, after a certain amount of time $T$ passed, skew bounds have converged to a value depending on the length of the path only. It turns out that skews on such a path may remain large for $\Omega(T)$ time. For that reason, we simply express the dynamic skew constraint in terms of the distance between two nodes in the $T$-stable subgraph of $G$.

**Definition 5.5** (Stable Gradient Skew and Stabilization Time)**.** *We say that Algorithm $\mathcal{A}$ exhibits a* stable gradient skew of $\mathcal{S}(\mathcal{G},\rho,U):\mathbb{R}^+\to\mathbb{R}^+$ with stabilization time $T$*, if for any execution of $\mathcal{A}$ with global skew $\mathcal{G}$, drift $\rho$, and uncertainty $U$ it holds that*

$$\forall v,w\in V,t\in\mathbb{R}_0^+:\ |L_v(t)-L_w(t)|\leq\mathcal{S}\left(d_T(v,w)\right).$$

*We will omit the arguments $\mathcal{G}$, $\rho$, and $U$ of $\mathcal{S}$ from the notation.*

Of particular interest is the *stable local skew* $\mathcal{S}(1)$ of an algorithm, as any two nodes that have been neighbors for $T$ time are guaranteed to have logical clock values that are merely $\mathcal{S}(1)$ off, which for the algorithm we are going to propose is typically small compared to $\mathcal{G}$.

## 5.2 Overview

We will start by describing a simple technique to obtain a global skew of

$$\mathcal{G}:=(1-\rho)(\mathcal{J}_D+1)+2\rho\left(D+2+\frac{\Lambda}{1+\rho}\right)$$

for arbitrary $\Lambda>0$. From results presented in [60], we will infer that this bound is essentially optimal if algorithms are required to guarantee the best possible *envelope condition*

$$\forall v\in V,t\in\mathbb{R}_0^+:\ |L_v(t)-t|\leq\rho t,$$

i.e., logical clocks do not differ further from the real time $t$ than the hardware clocks do in the worst case. Note that even if this condition is dropped, one cannot reduce the global skew by more than roughly a factor of two [16]. Observe that for maintaining a small global skew it is sufficient that nodes with the largest logical clock value in the system do not run faster than nodes with the smallest clock value whenever skews become "critical". Thus, it is not surprising that the respective rules never conflict with our goal to ensure a strong gradient property.

Subsequently, we expound our gradient clock synchronization algorithm $\mathcal{A}_\mu$. Given $\mathcal{G}$, $\rho$, and $U$, for any constant $\mu\geq 4\rho/(1-\rho)$ it guarantees a stable gradient skew of

$$\mathcal{S}_\mu(d)\in\Theta\left(Ud\log_{\mu/\rho}\left(\frac{\mathcal{G}}{Ud}\right)\right)$$

with stabilization time $T_\mu \in \Theta(\mathcal{G}/\mu)$. In fact, any $\mu$ larger than the minimum of $2\rho/(1-\rho)$ will do. However, the base of the logarithm in the skew bound tends to 1 as $\mu$ approaches $2\rho/(1-\rho)$ and the stabilization time deteriorates like the function $f(x) = 1/x$ for $x \to 0$.

Thus, for $\mu \in \Theta(\rho)$, we have a large stabilization time of $\Theta(\mathcal{G}/\rho)$ and a constant base, but in turn the logical clocks mimic the behavior of the hardware clocks with a slightly worse drift of $\mu + (1+\mu)\rho \in \Theta(\rho)$. On the other hand, choosing $\mu \in \Theta(1)$ leads to a stabilization time of $\Theta(\mathcal{G})$ and a large base of the logarithm. Considering that typically $D$ is smaller than $1/\rho$ (and certainly, say, $\rho^{-5}$), the latter choice in practice implies a local skew that is at most a constant multiple of $U$. Moreover, if the longest path in $G_{T_\mu}$ has a length of $\mathcal{O}(D)$, i.e., the diameter of $G$ is not small because of many short, but unreliable paths, we obtain a stronger bound on the maximum clock skew. Slightly abusing notation,[2] we show that $\mathcal{G} \in \mathcal{O}(UD)$ and thus

$$\mathcal{S}_\mu(d) \in \mathcal{O}\left(Ud \log_{\mu/\rho}\left(\frac{D}{d}\right)\right).$$

In other words, the rules employed to keep the local skew small by themselves impose a linear bound on the global skew. In a highly dynamic setting, however, the flooding technique presented in Section 5.3 is more reliable, as it does not build on the existence of (short) stable paths.

The above gradient property is asymptotically optimal as already in static graphs one can enforce comparable skews [60]. Indeed, $\mathcal{A}_\mu$ is $(1+o(1))$-competitive[3] in this regard. It is important to mention that a second lower bound, also given in [60], proves that $\mu \in \omega(1)$ is of no avail when trying to obtain a stronger gradient property. Intuitively, this arises from the fact that increasing clock speeds by more than a constant factor "amplifies" clock drifts, allowing an adversary to introduce skew into the system faster. Since the speed at which information spreads remains the same, this effect negates the advantage of being able to reduce clock skews more quickly.

Given these tight constraints on the stable skew, the stabilization time of $\mathcal{A}_\mu$ is asymptotically optimal as well [50], a statement that follows from a lower bound by Kuhn et al. [52]. Again, due to the limited speed of information propagation, $\mu \in \omega(1)$ is not sufficient to achieve a stabilization time of $o(\mathcal{G})$. In general, in order to allow for a faster integration of new edges, one must also accept a weaker gradient property (cf. [52]). If edge failures are short-lived, however, there is hope to reintegrate the respective edges sooner. A second algorithm presented in [50] achieves the same stable skew, but uses a different technique than $\mathcal{A}_\mu$ to stabilize new edges. This

[2] We demanded that $\mathcal{G}$ is a function of $D$, $\mathcal{J}_D$, and $\rho$ only. The second bound instead takes the maximal diameter of $G_{T_\mu}$, $U$, and $\rho$ as arguments.

[3] Here we refer to asymptotics with respect to fixed $U$ and $(\rho, \mathcal{S}_\mu(1)) \to (0, \infty)$.

algorithm gradually increases the relevance of skew observed on a given edge for the computation of the current logical clock rate. As a temporary failure of an edge does not mean that the stronger synchronization guarantees due to that edge are lost instantaneously, one could slowly decrease the relevance of the edge until it reappears and then start to increase it again.

## 5.3 Bounding the Global Skew

Ensuring an (almost) optimal worst-case global skew is surprisingly simple. Essentially, one cannot do better than the uncertainty of information on clock values between the most far apart nodes. Even if one node of such a pair tries to keep its clock exactly in the center of the interval in which the other must be according to its best guess, still a global skew of $\Omega(\mathcal{J}_D + \rho D)$ cannot be avoided. However, if a node runs fast without *knowing* that somebody is ahead, this might be a bad choice, as we might unnecessarily lose accuracy offered by the hardware clocks.

**Definition 5.6** (Time Envelope). *An algorithm satisfies the* envelope condition *provided that*

$$\forall v \in V, t \in \mathbb{R}_0^+ : \ |L_v(t) - t| \leq \rho t$$

*for any execution on any graph.*

Intuitively, this means that the system maintains the best possible worst-case approximation to real time that the hardware clocks permit. For algorithms satisfying this condition, the following stronger lower bound holds.

**Corollary 5.7.** *For clock synchronization algorithms for which the envelope condition holds, the global skew is at least*

$$\mathcal{G} \geq (1-\rho)\mathcal{J}_D + 2\rho D.$$

*If Condition* (5.1) *is dropped, i.e., clock rates are unrestricted, we still have*

$$\mathcal{G} \geq (1-\rho)\mathcal{J}_D + 2\rho \left\lfloor \frac{D}{2} \right\rfloor.$$

*These bounds hold also if $\rho$, $\mathcal{J}_D$, and $D$ are known to the algorithm and the graph is static.*

*Proof Sketch.* Suppose that $U \geq 1 + \rho$, i.e., for two nodes in distance $D$, the estimate obtained by a flooding is always the most accurate source of information about each other's clock values.

For this setting, in [60] it is shown that in a certain execution a skew of $(1-\rho)\mathcal{J}_D$ can be built up and maintained indefinitely. As soon as this skew is attained, all nodes have hardware and logical clock rate $1-\rho$. If we change the hardware clock rate of the node $v_{\max}$ with the largest clock value to $1+\rho$, we can make sure that it takes at least $D$ time until the node $v_{\min}$ with the smallest clock value learns about this and may increase its logical clock rate in response. In this time, the skew grows by $2\rho D$.

If decreasing logical clock rates is permitted, we make all nodes' clocks up to distance $\lfloor D/2 \rfloor$ from $v_{\max}$ (with respect to the flooding[4]) run fast, such that both $v_{\max}$ and $v_{\min}$ do not learn about this for $\lfloor D/2 \rfloor$ time. This yields the second bound. □

We will now show how to almost match this bound. Each node $v \in V$ maintains a local estimate $\hat{L}_v^{\max}$ of the maximum clock value in the system, which satisfies the following definition.

**Definition 5.8** (Max Estimates). *Denote by $\hat{L}_v^{\max}(t)$ the* max estimate *of node $v \in V$ at time $t$, where $\hat{L}_v^{\max}(0) := 0$. We require the following properties.*

*(i)* $\frac{d}{dt}\hat{L}_v^{\max}(t) = h_v(t)$ *(except when it is set to a larger value, see below).*

*(ii) For some value $\Lambda \in \mathbb{R}^+$, each node initiates a flooding distributing $\hat{L}_v^{\max}(t)$ at any time $t$ when $\hat{L}_v^{\max}(t) = \hat{L} \in \Lambda\mathbb{N}_0$.*

*(iii) When receiving such a flooding message at time $t$ that contains value $\hat{L}$ and was sent at time $t_s$, $v$ sets*

$$\hat{L}_v^{\max}(t) := \max\left\{\hat{L}_v^{\max}(t), \hat{L} + (1+\rho)(t-\hat{t}_s)\right\},$$

*where $\hat{t}_s \leq t$ is such that $t - \hat{t}_s \geq t - (t_s + \mathcal{J}_D)$, i.e., $t - \hat{t}_s$ is at least the time span for which $v$ can be sure that the message was en route. If $\hat{L}_v^{\max}(t)$ was set to a value larger than $\hat{L} + \Lambda$ and $v$ has not yet participated in a flooding containing a value of at least the maximum multiple of $\Lambda$ exceeded by $\hat{L}_v^{\max}(t)$, $v$ distributes $\hat{L}_v^{\max}(t)$ by a flooding.*

If we want to satisfy the envelope condition, this simple method guarantees an almost optimal estimator.

**Lemma 5.9.** *Suppose an algorithm satisfies Definition 5.8 and define*

$$\hat{L}^{\max}(t) := \max_{v \in V}\left\{\hat{L}_v^{\max}(t), \max_{\substack{v \text{ flooded at } t_s \\ \text{flooding not} \\ \text{complete at } t}}\left\{\hat{L}_v^{\max}(t_s) + (1+\rho)(t-t_s)\right\}\right\}.$$

*Then it holds that*

[4]In general this is not clearly defined because we did not specify the flooding mechanism and the graph changes dynamically. For the example of BFS flooding and a static graph, however, it is.

*(i)* $\forall v \in V, t \in \mathbb{R}_0^+ : \ \hat{L}_v^{\max}(t) \geq (1-\rho)t$

*(ii)* $\forall t_1 \leq t_2 \in \mathbb{R}_0^+ : \ \hat{L}^{\max}(t_2) - \hat{L}^{\max}(t_1) \leq (1+\rho)(t_2 - t_1)$

*(iii)* $\forall v \in V, t \in \mathbb{R}_0^+ :$
$\hat{L}_v^{\max}(t) > \hat{L}^{\max}(t) - (1-\rho)(\mathcal{J}_D + 1) - 2\rho\left(D + 2 + \frac{\Lambda}{1+\rho}\right).$

*Proof.* Property $(i)$ immediately follows from Property $(i)$ in Definition 5.8, the minimum hardware clock speed, and the fact that the estimates are never decreased. Since the $\hat{L}_v^{\max}$ change only finitely often in finite time, they are differentiable at all but countably many points with $\frac{d}{dt}\hat{L}_v^{\max}(t) = h_v(t)$. By Theorem 2.23, this implies that $\frac{d}{dt}\hat{L}^{\max}(t)$ exists at all but countably many points and is bounded by $1+\rho$. Hence, for any interval $(t_1, t_2]$ during which $L^{\max}$ is continuous, we have

$$L^{\max}(t_2) = L^{\max}(t_1) + \int_{t_1}^{t_2} \frac{d}{dt} L^{\max}(\tau)\ d\tau \leq L^{\max}(t_1) + (1+\rho)(t_2 - t_1).$$

On the other hand, observe that when a value increases according to Property $(iii)$ from Definition 5.8, it becomes at most

$$\hat{L} + (1+\rho)(t - \hat{t}_s) \leq \hat{L}^{\max}(t_s) + (1+\rho)(t - t_s).$$

As $\hat{L}^{\max}$ may only drop at discontinuities, we conclude that Property $(ii)$ is satisfied.

Regarding Property $(iii)$, observe that Properties $(i)$ and $(ii)$ show the statement for times $t < D+1$, i.e., we may w.l.o.g. assume that $t \geq D+1$. Thus, any node $v \in V$ has received at least one flooding message. Denote by $\hat{L} \in \Lambda\mathbb{N}$ the largest multiple of $\Lambda$ such that for the time $t_{\hat{L}}$ with $\hat{L}^{\max}(t_{\hat{L}}) = \hat{L}$ it holds that

$$t_{\hat{L}} \leq t - (D+1).$$

Similarly, denote by $t_{\hat{L}+\Lambda}$ the time when $\hat{L}^{\max}(t_{\hat{L}+\Lambda}) = \hat{L} + \Lambda$. From Property $(ii)$, we get that

$$\hat{L}^{\max}(t) \leq \hat{L} + \Lambda + (1+\rho)(t - t_{\hat{L}+\Lambda}).$$

By definition of $\hat{L}^{\max}$ and Properties $(ii)$ and $(iii)$ of the max estimates, there must be some node $w$ initiating a flooding containing $\hat{L}$ at a time $t_s \geq t_{\hat{L}}$, either because $\hat{L}_w^{\max}(t_{\hat{L}}) = \hat{L}$ or it receives a message from an ongoing flooding causing it to set $\hat{L}_w^{\max}(t_s)$ to a value of at least $\hat{L} + (1+\rho)(t_s - t_{\hat{L}})$. Note that $v$ receives a respective flooding message at a time $t_r \leq t$, as the definition of the dynamic diameter and the normalization of message delays imply that

$$t_r \leq t_s + D \leq t_{\hat{L}} + D + 1 \leq t.$$

As delays are at most one, the estimate $v$ receives at time $t_r$ is at least $\hat{L}+(1+\rho)(t_s-t_{\hat{L}}-1)$. Thus, by Properties (*i*) and (*iii*) of the max estimates, we have that

$$\hat{L}_v^{\max}(t) \geq \hat{L} + (1+\rho)(\max\{t_r - (t_{\hat{L}} + \mathcal{J}_D), 0\} - 1) + (1-\rho)(t-t_r).$$

Observe that decreasing $t_r$ below $t_{\hat{L}} + \mathcal{J}_D$ will only increase $\hat{L}_v^{\max}(t)$ due to the term $(1-\rho)(t-t_r)$, without affecting the bound on $\hat{L}^{\max}(t)$. Thus, w.l.o.g., $t_r \geq t_{\hat{L}} + \mathcal{J}_D$ and hence

$$\hat{L}_v^{\max}(t) \geq \hat{L} + (1+\rho)(t_r - (t_{\hat{L}} + \mathcal{J}_D) - 1) + (1-\rho)(t-t_r).$$

Moreover, we have that $t < t_{\hat{L}+\Lambda} + D + 1$ by the definitions of $D$, $t_r$ and $t_{\hat{L}+\Lambda}$. Combining these bounds with the above inequalities yields

$$\begin{aligned}
& \hat{L}^{\max}(t) - \hat{L}_v^{\max}(t) \\
\leq\ & \Lambda + (1+\rho)(\mathcal{J}_D + 1 - (t_{\hat{L}+\Lambda} - t_{\hat{L}})) + 2\rho(t-t_r) \\
<\ & \Lambda + (1+\rho)(\mathcal{J}_D + 1 - (t_{\hat{L}+\Lambda} - t_{\hat{L}})) + 2\rho(t_{\hat{L}+\Lambda} + D + 1 - (t_{\hat{L}} + \mathcal{J}_D)) \\
=\ & \Lambda - (1+\rho)(t_{\hat{L}+\Lambda} - t_{\hat{L}}) + (1-\rho)(\mathcal{J}_D + 1) + 2\rho(D + 2 + t_{\hat{L}+\Lambda} - t_{\hat{L}}) \\
\overset{(ii)}{\leq}\ & (1-\rho)(\mathcal{J}_D + 1) + 2\rho\left(D + 2 + \frac{\Lambda}{1+\rho}\right).
\end{aligned}$$

Since $v$ and $t \geq D+1$ were arbitrary, this shows Property (*iii*), concluding the proof. $\square$

Property (*iii*) of the max estimates shown in this lemma gives rise to a very simple strategy to get arbitrary close to the lower bound from Corollary 5.7. If an algorithm makes sure that any node $v \in V$ with $L_v(t) = \hat{L}_v^{\max}(t)$ is slow, this guarantees that $\max_{v\in V}\{L_v(t)\} \leq \hat{L}^{\max}(t)$ at all times. Thus, any node whose logical clock falls by $(1-\rho)(\mathcal{J}_D+1) + 2\rho(D+2+\Lambda/(1+\rho))$ behind $\max_{v\in V}\{L_v(t)\}$ will notice that $L_v(t) < \hat{L}_v^{\max}(t)$ and can increase its clock speed to avoid larger skews. This is formalized as follows.

**Definition 5.10** (Max Estimate Algorithms)**.** *Suppose an algorithm satisfies Definition 5.8. It is a* max estimate algorithm *provided that, for all nodes $v \in V$ and times $t \in \mathbb{R}_0^+$, it holds that*

*(i)* $L_v(t) = \hat{L}_v^{\max}(t) \Rightarrow l_v(t) = h_v(t)$

*(ii)* $L_v(t) < \hat{L}_v^{\max}(t) \Rightarrow l_v(t) \geq \frac{1+\rho}{1-\rho} h_v(t)$.

**Theorem 5.11.** *Any max estimate algorithm has a global skew of*

$$\mathcal{G} := (1-\rho)(\mathcal{J}_D + 1) + 2\rho\left(D + 2 + \frac{\Lambda}{1+\rho}\right)$$

*and satisfies the envelope condition.*

*Proof.* Due to Property (*i*) from Definition 5.10, the fact that max estimates never decrease, and Property (*i*) from Definition 5.8, we have that $\max_{v \in V}\{L_v(t)\}$ is bounded by $\hat{L}^{\max}(t)$. Due to Property (*ii*) from Lemma 5.9 (for $t_1 = 0$ and $t_2 = t$) and the fact that $l_v(t) \geq h_v(t)$ for all nodes $v \in V$ and times $t$, this shows that the algorithm satisfies the envelope condition.

Moreover, using the notation from Lemma 5.9, Theorem 2.23 yields that

$$g(t) := \hat{L}^{\max}(t) - \min_{v \in V}\{L_v(t)\} = \hat{L}^{\max}(t) + \max_{v \in V}\{-L_v(t)\}$$

is differentiable at all but countably many points with

$$\frac{d}{dt}g(t) \quad \leq \quad 1 + \rho - \min_{\substack{v \in V \\ \hat{L}^{\max}(t) - L_v(t) = g(t)}} \{l_v(t)\}.$$

At any time when $g(t) \geq \mathcal{G}$, Property (*iii*) from Lemma 5.9 gives for all $v \in V$ for which $\hat{L}^{\max}(t) - L_v(t) = g(t) \geq \mathcal{G}$ that $L_v(t) < L_v^{\max}(t)$. Thus, Property (*ii*) from Definition 5.10 yields that for any such $v$ we have $l_v(t) \geq (1+\rho)h_v(t)/(1-\rho) \geq 1 + \rho$. It follows that at any time $t$ when $g(t) \geq \mathcal{G}$ and the derivative exists, we have $\frac{d}{dt}g(t) \leq 0$. Note that $g$ can only be discontinuous due to $\hat{L}^{\max}$, which due to Property (*ii*) from Lemma 5.9 cannot increase instantaneously. Thus, as $g(0) = 0$, we must have $g(t) \leq \mathcal{G}$ at all times. Recalling that $L_v(t) \leq \hat{L}^{\max}(t)$ for all $v \in V$ and $t$, we see that indeed for all $v, w \in V$ and $t \in \mathbb{R}_0^+$ it holds that $|L_v(t) - L_w(t)| \leq \mathcal{G}$ as claimed. □

Together with Corollary 5.7, this theorem states that for the class of algorithms satisfying the envelope condition the given technique is optimal. In particular, the classical algorithm which—roughly speaking—simply outputs $L_v(t) := \hat{L}_v^{\max}(t)$ as its logical clock value [99] achieves an optimal global skew under the constraints that the envelope condition must hold and logical clocks are never slowed down in comparison to hardware clocks. However, this algorithm has two shortcomings: Neither does it exhibit a (non-trivial) gradient property nor does it provide any upper bound on logical clock rates.

Theorem 5.11 demonstrates that the latter can be achieved by an algorithm that is uniform with respect to all model parameters but $\rho$. We can maintain exactly the same skew bounds by just slightly increasing clock speeds. We remark that speeding up clocks by less than factor $(1+\rho)/(1-\rho)$ makes it impossible to compensate for the hardware clock drift, i.e., the range $[1-\rho, (1+\rho)^2/(1-\rho)]$ for logical clock rates is also best possible.

Moreover, due to the following reasons the technique does not conflict with our approach to achieve an optimal gradient property:

- If the Algorithm $\mathcal{A}_\mu$ presented in the next section requires logical clocks to be fast, this is due to a neighbor whose clock is certainly ahead. Similarly, if it requires clocks to be slow, this is due to a neighbor whose clock is certainly behind or because the nodes clock value attains the current maximum.
- Property (*iii*) from Definition 5.8 can be changed in that any node $v \in V$ also increases $\hat{L}_v^{\max}$ if it learns that a neighbor has a larger clock value. This does not change the properties of $\hat{L}_v^{\max}$ and $\hat{L}^{\max}$ that we have shown, yet ensures that nodes will not increase their logical clocks beyond $\hat{L}_v^{\max}$ because of the rules of $\mathcal{A}_\mu$.
- To prove Theorem 5.11, it is sufficient that nodes whose clocks attain the minimum clock value in the network increase their logical clocks faster than their hardware clock rate. Hence, the bound on the global skew still holds if nodes are required to be slow because of some neighbor that lags behind.

To simplify the presentation, in the following we assume that a global skew of $\mathcal{G}$ as given by Theorem 5.11 is guaranteed. The reader who would like to see a combined algorithm minimizing both global and local skew is referred to [60, 71]; the dynamic setting does not pose new challenges in this regard.

## 5.4 An Algorithm with Optimal Gradient Property

In this section, we will present Algorithm $\mathcal{A}_\mu$, which is an abstract variant of the algorithm from [51]. We will discuss how to adapt the algorithm and its analysis to a more realistic model in Section 5.6.

Algorithm $\mathcal{A}_\mu$ operates in lock-step rounds of duration $\Theta(\mathcal{G}/\mu)$, where a global skew of $\mathcal{G}$ is ensured by some separate mechanism. By $T_1^{(r)}$ we denote the time when round $r \in \mathbb{N}$ begins, i.e., $T_1^{(1)} = 0$. Each node $v \in V$ maintains for each $s \in \mathbb{N}$ a dynamic subset of neighbors $\mathcal{N}_v^s(t) \subseteq \mathcal{N}_v(t)$, initialized to $\mathcal{N}_v(0)$. During each round, edges that have newly appeared until the beginning of the round are gradually incorporated into the algorithm's decisions. To this end, each node sets $\mathcal{N}_v^1(T_1^{(r)}) := \mathcal{N}_v(T_1^{(r)})$ at the beginning of each round $r \geq 2$. Similarly, for $s \geq 2$ and each $r \in \mathbb{N}$, each node sets $\mathcal{N}_v^s(T_s^{(r)}) := \mathcal{N}_v^{s-1}(T_s^{(r)})$ at the times

$$T_s^{(r)} := T_1^{(r)} + \sum_{s'=1}^{s-1} \frac{\mathcal{G}}{\sigma^{s'-1}(1-\rho)\mu}, \tag{5.2}$$

where

$$\sigma := \frac{(1-\rho)\mu}{2\rho} > 1. \tag{5.3}$$

On the contrary, whenever an edge $\{v,w\}$ disappears, node $w$ is removed from *all* sets $\mathcal{N}_v^s$, $s \in \mathbb{N}$ (and vice versa). For the sake of a concise notation, however, we define that $w$ is still present in $\mathcal{N}_v(t)$ and $\mathcal{N}_v^s(t)$, $s \in \mathbb{N}$, respectively (in contrast to the convention that if a variable changes at time $t$ it already attains the new value at time $t$).

We set the duration of a whole round to

$$\frac{T_\mu}{2} := \frac{\mathcal{G}}{(1-\rho)\mu}\left(1+\sum_{s'=1}^{\infty}\sigma^{-s'+1}\right) = \frac{(2\sigma-1)\mathcal{G}}{(\sigma-1)(1-\rho)\mu}, \tag{5.4}$$

i.e., $T_1^{(r)} := (r-1)T_\mu/2$. Note that while appealingly simple, this scheme cannot be implemented in practice, as it requires that all nodes *synchronously* modify their sets at specific real times; this will be discussed in Section 5.6.

From these sets and the estimated clock values of its neighbors, $v$ determines its logical clock rate. More precisely, for a value $\kappa > 2U$, $\mathcal{A}_\mu$ satisfies the following conditions.

**Definition 5.12** (Fast Condition)**.** *The* fast condition on level $s \in \mathbb{N}$ *states that for all nodes $v \in V$ and times $t \in \mathbb{R}_0^+$ we have*

$$\left.\begin{array}{l}\exists w \in \mathcal{N}_v^s(t):\ L_w(t)-L_v(t) \geq s\kappa \\ \forall u \in \mathcal{N}_v^s(t):\ L_v(t)-L_u(t) \leq s\kappa\end{array}\right\} \Rightarrow l_v(t) = (1+\mu)h_v(t).$$

Informally, the fast condition accomplishes the following. If for some $v \in V$ and $s \in \mathbb{N}$ node $w \in V$ maximizes the expression $L_v - L_w - d_s(v,w)s\kappa$ (where $d_s(v,w)$ denotes the distance of $v$ and $w$ in the graph induced by the neighborhoods $\mathcal{N}_u^s$, $u \in V$), then $w$ is fast, as otherwise for one of its neighbors this function would attain an even larger value. Thus, nodes which fall too far behind will catch up, granted that the nodes with large clock values are slow. Ensuring the latter is the goal of the second condition, which enforces that if for some $v \in V$ and $s \in \mathbb{N}$ node $w \in V$ maximizes the expression $L_w - L_v - d_s(v,w)(s+1/2)\kappa$, then $w$ is slow.

**Definition 5.13** (Slow Condition)**.** *The* slow condition on level $s \in \mathbb{N}$ *states that for all nodes $v \in V$ and times $t \in \mathbb{R}_0^+$ we have*

$$\left.\begin{array}{l}\forall w \in \mathcal{N}_v^s(t):\ L_w(t)-L_v(t) \leq \left(s+\frac{1}{2}\right)\kappa+\delta \\ \exists u \in \mathcal{N}_v^s(t):\ L_v(t)-L_u(t) \geq \left(s+\frac{1}{2}\right)\kappa-\delta\end{array}\right\} \Rightarrow l_v(t) = h_v(t),$$

*where $\delta \in \mathbb{R}^+$ is arbitrarily small.*

If neither of the conditions hold at node $v \in V$, its logical clock may run at any rate from the range $[h_v, (1+\mu)h_v]$, with the following exception.

**Definition 5.14** (Controlled Maximum Clock). *If for any node $v \in V$ it holds that $L_v(t) = \max_{w\in V}\{L_w(t)\}$ at a time $t \in \mathbb{R}_0^+$, then $l_v(t) = h_v(t)$.*

We point out that we introduce this condition mainly for convenience reasons, as the fast and slow conditions alone are sufficient to guarantee a strong gradient property. Nevertheless, in order to ensure a good global skew it is desirable that Definition 5.14 or a similar constraint is satisfied by the algorithm.

The requirement that $\kappa > 2U$ originates from the fact that the fast and slow conditions must not be contradictory. For this it is not enough that the preconditions for a node being fast respectively slow are mutually exclusive. As the nodes have only access to estimates of their neighbors' clock values that may differ by up to $U$ from the actual values, in case $\kappa/2 \leq U$ there are indistinguishable executions where either the fast or the slow condition hold at a node. Thus, $\kappa > 2U$ is necessary for an algorithm to realize these rules in any circumstance. As the proof that $\kappa > 2U$ is indeed sufficient to implement all conditions concurrently is a technicality, we omit it here and refer to [51].

Intuitively, the fast and the slow condition on level $s \in \mathbb{N}$ work together as follows. If on some path an adversary tries to accumulate clock skew beyond an average of $(s + 1/2)\kappa$ per hop that is not already present somewhere in the graph, this can only happen at rate $2\rho$ due to the slow condition. On the other hand, this means that there must be much skew exceeding an average of $s\kappa$, which is reduced at a rate of at least $(1 + \mu)(1 - \rho) - (1 + \rho) \in \Omega(\mu)$ due to the fast condition. Hence, whatever the length of the longest possible path of average skew $(s + 1/2)\kappa$ is, the longest path with an average skew that is by $\kappa$ larger will be shorter by factor $\Theta(\mu/\rho)$. The next section deals with formalizing and proving this statement.

## 5.5 Analysis of Algorithm $\mathcal{A}_\mu$

Before we can start our analysis, we need to introduce some definitions. In order to establish the gradient property on edges that appeared recently, Algorithm $\mathcal{A}_\mu$ sequentially activates the slow and fast conditions starting from the lowest level. The following definition introduces the notions to capture the subgraphs we need to consider at a given time for a given level.

**Definition 5.15** (Level-$s$ Edges and Paths). *For $s \in \mathbb{N}$, we say that edge $\{v, w\} \in \binom{V}{2}$ is a* level-$s$ edge at time $t \in \mathbb{R}_0^+$ *if $w \in \mathcal{N}_v^s(t)$ (and vice versa). We define $E^s(t)$ to be the set of level-$s$ edges at time $t$. For any path $p$, denote by $E_p$ the set of its edges and by $\ell_p$ its length. We define for $s \in \mathbb{N}$ and times $t \in \mathbb{R}_0^+$ the set of* level-$s$ paths at time $t$ *as*

$$P^s(t) := \{\textit{path } p \,|\, E_p \subseteq E^s(t)\}.$$

*Moreover, for each $v \in V$, the set of* level-$s$ paths starting at node $v$ at time *$t$ is*

$$P_v^s(t) := \{\mathit{path}\ p = (v, \ldots) \,|\, E_p \subseteq E^s(t)\}.$$

Roughly speaking, within a certain range of skews the algorithm is proactive and quickly reacts to clock skews. For each level $s \in \mathbb{N}$, the crucial magnitude is the skew on a path that exceeds an average of $s\kappa$, motivating the following definition.

**Definition 5.16** (Catch-Up Potential)**.** *For all paths $p = (v, \ldots, w)$, $s \in \mathbb{N}$, and times $t \in \mathbb{R}_0^+$, we define*

$$\xi_p^s(t) := L_v(t) - L_w(t) - s\kappa\ell_p.$$

*Moreover, for each $v \in V$,*

$$\Xi_v^s(t) := \max_{p \in P_v^s(t)} \{\xi_p^s(t)\}.$$

On the other hand, $\mathcal{A}_\mu$ makes sure to not always act rashly, as otherwise the uncertainty in clock estimates could lead to all nodes being fast, inhibiting their ability to reduce skews. This time, for each level $s \in \mathbb{N}_0$, the decisive value is the skew on a path exceeding an average of $(s + 1/2)\kappa$.

**Definition 5.17** (Runaway Potential)**.** *For all paths $p = (v, \ldots, w)$, $s \in \mathbb{N}_0$, and times $t \in \mathbb{R}_0^+$, we define*

$$\psi_p^s(t) := L_v(t) - L_w(t) - \left(s + \frac{1}{2}\right)\kappa\ell_p.$$

*Moreover,*

$$\Psi^s(t) := \max_{p \in P^s(t)} \{\psi_p^s(t)\}.$$

Essentially, we are going to show that the maximal length of a path with an average skew of at least $(s + 1/2)\kappa$ decreases exponentially in $s$, leading to a logarithmic skew bound between neighbors. However, the algorithm perpetually adds edges on the various levels, which may also affect the skew bounds on long-standing paths. In order to reflect this in our analysis, we need to argue more generally, necessitating the following definition.

**Definition 5.18** (Gradient Sequences)**.** *A non-increasing sequence of positive Reals $C = \{C_s\}_{s \in \mathbb{N}_0}$ is a* gradient sequence *if $\kappa C_0 \geq \mathcal{G}$.*

Depending on the considered gradient sequence, we can now formulate the condition under which the network is in a valid state.

**Definition 5.19** (Legality)**.** *Given a weighted, dynamic graph $G$ and a gradient sequence $C$, for each $s \in \mathbb{N}_0$ the system is* $s$-legal with respect to $C$ at time $t \in \mathbb{R}_0^+$, *if and only if it holds that*

$$\Psi^s(t) \leq \kappa C_s.$$

*The system is* $C$-legal at time $t$ *if it is $s$-legal for all $s \in \mathbb{N}$ at time $t$ with respect to $C$.*

The goal of our analysis is to prove that at all times the network remains in a legal state with respect to a certain gradient sequence whose values decrease—starting from the second level—exponentially. Some basic observations can immediately be deduced from the given definitions.

**Lemma 5.20.** *The following statements hold at all times $t \in \mathbb{R}_0^+$.*

*(i) The system is $0$-legal.*

*(ii) If for some $s \in \mathbb{N}_0$ the system is $s$-legal and $C_{s+1} = C_s$, then the system is also $(s+1)$-legal.*

*Proof.* Statement $(i)$ holds by definition as for any path $p = (v, \ldots, w)$ and any time $t$ we have

$$\psi_p^0(t) \leq |L_v(t) - L_w(t)| \leq \mathcal{G} \leq \kappa C_0.$$

For statement $(ii)$, recall that Algorithm $\mathcal{A}_\mu$ ensures at all times $t$ and nodes $v \in V$ that $\mathcal{N}_v^{s+1}(t) \subseteq \mathcal{N}_v^s(t)$, i.e., $E^{s+1}(t) \subseteq E^s(t)$. Hence, $\Psi^{s+1}(t) \leq \Psi^s(t)$, which due to the assumptions of $s$-legality and $C_s = C_{s+1}$ yields the claim. □

The first property will serve as an induction anchor. Legality on higher levels then will follow from the bounds on the previous level. The second statement basically shows that we can add new edges level-wise, as this means that even if we "tamper" with the state on level $s$ by inserting new edges, the inductive chain proving the skew bounds on higher levels can be maintained.

We now prove the first key lemma of our analysis, which states that the nodes maximizing $\Xi_v^s$ must be fast whenever $\Xi_v^s > 0$. This permits to bound the rate at which $\Xi_v^s$ changes in terms of $l_v$ at all times except $T_s^{(r)}$, $r \in \mathbb{N}$. At these times the algorithm inserts edges on level $s$, potentially reducing distances in the graph induced by the level-$s$ edges, maybe increasing $\Xi_v^s$.

**Lemma 5.21.** *Suppose for a node $v \in V$, some $s \in \mathbb{N}$, and a time interval $(t_0, t_1]$ such that $T_s^{(r)} \notin (t_0, t_1] \subset \mathbb{R}_0^+$ for all $r \in \mathbb{N}$ it holds that $\Xi_v^s(t) > 0$ for all $t \in (t_0, t_1)$. Then we have that*

$$\Xi_v^s(t_1) - \Xi_v^s(t_0) \leq L_v(t_1) - L_v(t_0) - (1+\mu)(1-\rho)(t_1 - t_0).$$

*Proof.* For a time $t \in (t_0, t_1]$, consider any path $p = (v, \dots, w', w)$ maximizing $\Xi_v^s(t)$, i.e., $\xi_p^s(t) = \Xi_v^s(t)$. It holds that

$$L_{w'}(t) - L_w(t) \geq s\kappa,$$

as otherwise we must have $\xi^s_{(v,\dots,w')}(t) > \xi_p^s(t)$. Similarly, for all $u \in \mathcal{N}_w^s(t)$ it holds that

$$L_w(t) - L_u(t) \geq s\kappa,$$

as otherwise we must have $\xi^s_{(v,\dots,w,u)}(t) > \xi_p^s(t)$. Hence, according to the fast condition, we have $l_w(t) = (1+\mu)h_v(t) \geq (1+\mu)(1-\rho)$. By Theorem 2.23, $\Xi_v^s$ is thus differentiable at all but countably many points with derivative bounded by $l_v(t) - (1+\mu)(1-\rho)$.

Because the algorithm adds edges on level $s$ at the times $T_s^{(r)} \notin (t_0, t_1]$, paths can only be removed from $P_v^s(t)$ during $(t_0, t_1]$, i.e., $\Xi_v^s$ may only drop at discontinuities. We conclude that

$$\Xi_v^s(t_1) - \Xi_v^s(t_0) \leq L_v(t_1) - L_v(t_0) - (1+\mu)(1-\rho)(t_1 - t_0)$$

as claimed. □

We will need a helper statement showing that under certain circumstances the preconditions of the previous lemma are always met.

**Lemma 5.22.** *Assume that for some node $v \in V$ and $s \in \mathbb{N}$, we have for all $r \in \mathbb{N}$ that $T_s^{(r)} \notin (t_0, t_1] \subset \mathbb{R}_0^+$. Suppose that*

$$L_v(t_1) - L_v(t_0) \leq (1+\rho)(t_1 - t_0)$$

*and that*

$$\Xi_v^s(t_1) > 2\rho(t_1 - t_0).$$

*Then*

$$\forall t \in [t_0, t_1] : \Xi_v^s(t) > 0.$$

*Proof.* Let $p = (v, \dots, w) \in P^s(t_1)$ maximize $\Xi_v^s(t_1)$, i.e., $\Xi_v^s(t_1) = \xi_p^s(t_1)$. Since no edges are added on level $s$ during $(t_0, t_1]$, we have that $p \in P^s(t)$ for all $t \in [t_0, t_1]$. Thus, we can bound

$$\begin{array}{rcl}
\Xi_v^s(t) & \geq & \xi_p^s(t) \\
& = & \xi_p^s(t_1) - (L_v(t_1) - L_v(t)) + L_w(t_1) - L_w(t) \\
& \geq & \xi_p^s(t_1) - (L_v(t_1) - L_v(t_0)) + L_v(t) - L_v(t_0) + (1-\rho)(t_1 - t) \\
& \geq & \Xi_v^s(t_1) - 2\rho(t_1 - t_0) \\
& > & 0.
\end{array}$$

□

Before we can move on to the main lemma, we need to derive a technical result from the slow condition. Intuitively, we show that if a node's clock increased faster than at the maximum hardware clock rate during a given time period, the slow condition entails that there must have been a neighbor that was far ahead or all neighbors were close.

**Lemma 5.23.** *Assume that for some node $v \in V$ and $s \in \mathbb{N}$, we have for all $r \in \mathbb{N}$ that $T_s^{(r)} \notin (t_0, t_1] \subset \mathbb{R}_0^+$. If*

$$t_{\min} := \min\{t \in [t_0, t_1] \,|\, L_v(t_1) - L_v(t) \leq (1+\rho)(t_1 - t)\}$$

*is greater than $t_0$, then*

$$\exists w \in \mathcal{N}_v^s(t_{\min}) : \; L_w(t_{\min}) - L_v(t_{\min}) > \left(s + \frac{1}{2}\right)\kappa \tag{5.5}$$

*or*

$$\forall u \in \mathcal{N}_v^s(t_{\min}) : \; L_v(t_{\min}) - L_u(t_{\min}) < \left(s + \frac{1}{2}\right)\kappa. \tag{5.6}$$

*Proof.* Assuming the contrary, the logical negation of (5.5) $\vee$ (5.6) is

$$\begin{aligned} & \forall w \in \mathcal{N}_v^s(t_{\min}) : \; L_w(t_{\min}) - L_v(t_{\min}) \leq \left(s + \frac{1}{2}\right)\kappa \\ \wedge \quad & \exists u \in \mathcal{N}_v^s(t_{\min}) : \; L_v(t_{\min}) - L_u(t_{\min}) \geq \left(s + \frac{1}{2}\right)\kappa. \end{aligned}$$

As $T_s^{(r)} \notin (t_0, t_1]$ for all $r$, no neighbors are added to $\mathcal{N}_v^s$ in the time interval $(t_0, t_1]$. Because edges that are removed at time $t_{\min}$ are still in $\mathcal{N}_v^s(t_{\min})$ and $\mathcal{N}_v^s$ is finite at all times, there must be a closed time interval of non-zero length ending at time $t_{\min}$ during which $\mathcal{N}_v^s$ does not change. Thus, as logical clocks are continuous and $t_{\min} > t_0$, a time $t'_{\min} \in (t_0, t_{\min})$ exists such that for all $t \in [t'_{\min}, t_{\min}]$ it holds that

$$\begin{aligned} & \forall w \in \mathcal{N}_v^s(t_{\min}) = \mathcal{N}_v^s(t) : \; L_w(t) - L_v(t) \leq \left(s + \frac{1}{2}\right)\kappa + \delta \\ \wedge \quad & \exists u \in \mathcal{N}_v^s(t_{\min}) = \mathcal{N}_v^s(t) : \; L_v(t) - L_u(t) \geq \left(s + \frac{1}{2}\right)\kappa - \delta. \end{aligned}$$

Due to the slow condition, we get that $l_u(t) = h_v(t)$ for all $t \in [t'_{\min}, t_{\min}]$ and therefore

$$L_v(t_1) - L_v(t'_{\min}) = L_v(t_1) - L_v(t_{\min}) + L_v(t_{\min}) - L_v(t'_{\min}) \leq (1+\rho)(t_1 - t'_{\min}),$$

contradicting the minimality of $t_{\min}$. □

We are now ready to advance to the heart of the proof of the gradient property of $\mathcal{A}_\mu$. Combining the fast condition and the slow condition on level $s$ in form of the Lemmas 5.21 and 5.23, we show that if no edges are added on level $s$ and the system is $(s-1)$-legal for a certain amount of time, the system becomes legal on level $s$ with respect to $C_s \in \Omega(\rho C_{s-1}/\mu)$, i.e., we gain a factor of $\Theta(\mu/\rho)$.

Since the proof of the lemma is quite intricate, we sketch the main concepts first. The proof will comprise two parts. Assuming that the claim does not hold for some path $p$, we backtrack the reason of its violation in time. Starting from $p$, we will construct a sequence of paths that are to be held "responsible" for the large skews in the system, removing nodes from $p$ in the first part of the proof, then extending the path in the second. Whenever we switch from a path $p_i = (v, \ldots)$ to a path $p_{i+1}$, from Lemma 5.23 we get a positive lower bound on $\xi^s_{p_{i+1}}$ that is comparable to the one we had on $\xi^s_{p_i}$. In between, the respective node $v$ ran at an amortized rate of at most $(1+\rho)$, therefore $\Xi^s_v$ must have decreased at an (amortized) rate of at least $(1-\rho)\mu - (1+\rho)$ according to Lemma 5.21. Since we went back in time sufficiently far, this will finally lead to the conclusion that for some node $w \in V$, $\Xi^s_w \leq \Psi^{s-1}$ must have been overly large, contradicting the prerequisite that the system is $(s-1)$-legal during the considered time interval.

**Lemma 5.24.** *Assume that for some $s \in \mathbb{N}$ the system is $(s-1)$-legal during the time interval $\left[\underline{t}, \bar{t}\right] \subset \mathbb{R}^+_0$ and that for all $r \in \mathbb{N}$ we have $T^{(r)}_s \notin \left(\underline{t}, \bar{t}\right]$. Recall that*

$$\sigma = \frac{(1-\rho)\mu}{2\rho} > 1$$

*and define*

$$\nabla^s := \frac{\kappa C_{s-1}}{(1-\rho)\mu}. \tag{5.7}$$

*Then for all times $t \in \left[\underline{t} + \nabla^s, \bar{t}\right]$ it holds that*

$$\Psi^s(t) \leq 2\rho\nabla^s = \frac{\kappa C_{s-1}}{\sigma}.$$

*Proof.* Assume for contradiction that there is a time $t_0 \in \left[\underline{t} + \nabla^s, \bar{t}\right]$ with $\Psi^s(t_0) > \kappa C_{s-1}/\sigma$, i.e., there is a node $v_0 \in V$ and a path $p = (v_0, \ldots, v_k) \in P^s_{v_0}(t_0)$ with

$$\psi^s_p(t_0) = \Psi^s(t_0) > 2\rho\nabla^s. \tag{5.8}$$

**Part I. Getting closer to $v_k$.** We inductively define a sequence of decreasing times $t_0 \geq t_1 \geq \ldots \geq t_{l+1}$, where $t_{l+1} \geq t_0 - \nabla^s \geq \underline{t}$. Given $t_i$, we set

$$t_{i+1} := \min\{t \in [t_0 - \nabla^s, t_i] \mid L_{v_i}(t_i) - L_{v_i}(t) \leq (1+\rho)(t_i - t)\}. \tag{5.9}$$

For $i \leq k$ and $t_i \geq t_0 - \nabla^s$, time $t_{i+1}$ is well-defined. We halt the construction at index $l$ if $t_{l+1} = t_0 - \nabla^s$ or if

$$\exists w \in \mathcal{N}_{v_l}^s(t_{l+1}) : \; L_w(t_{l+1}) - L_{v_l}(t_{l+1}) > \left(s + \frac{1}{2}\right)\kappa. \tag{5.10}$$

We will see later that the construction can never reach node $v_k$.

If the construction does not halt at index $i$, Lemma 5.23 states that

$$\forall w \in \mathcal{N}_{v_i}^s(t_{i+1}) : \; L_{v_i}(t) - L_w(t) < \left(s + \frac{1}{2}\right)\kappa. \tag{5.11}$$

We show by induction that for all $i \in \{0, \ldots, l\}$ it holds that

$$\xi^s_{(v_i,\ldots,v_k)}(t_{i+1}) \geq \Psi^s(t_0) + \frac{(k-i)\kappa}{2} - (1+\rho)(t_0 - t_{i+1}) + L_{v_k}(t_0) - L_{v_k}(t_{i+1}). \tag{5.12}$$

For the base case we compute

$$\begin{aligned}
\xi^s_{(v_0,\ldots,v_k)}(t_1) &= \xi^s_p(t_0) - (L_{v_0}(t_0) - L_{v_0}(t_1)) + L_{v_k}(t_0) - L_{v_k}(t_1) \\
&\overset{(5.9)}{\geq} \xi^s_p(t_0) - (1+\rho)(t_0 - t_1) + L_{v_k}(t_0) - L_{v_k}(t_1) \\
&= \psi^s_p(t_0) + \frac{k\kappa}{2} - (1+\rho)(t_0 - t_1) + L_{v_k}(t_0) - L_{v_k}(t_1) \\
&\overset{(5.8)}{=} \Psi^s(t_0) + \frac{k\kappa}{2} - (1+\rho)(t_0 - t_1) + L_{v_k}(t_0) - L_{v_k}(t_1).
\end{aligned}$$

As for the induction step, assume that the claim holds for $i < l$. From Inequality (5.11) we know that

$$L_{v_i}(t_{i+1}) - L_{v_{i+1}}(t_{i+1}) < \left(s + \frac{1}{2}\right)\kappa. \tag{5.13}$$

Thus, we can write

$$\begin{aligned}
\xi^s_{(v_{i+1},\ldots,v_k)}(t_{i+1}) &= \xi^s_{(v_i,\ldots,v_k)}(t_{i+1}) - L_{v_i}(t_{i+1}) + L_{v_{i+1}}(t_{i+1}) + s\kappa \\
&\overset{(5.13)}{>} \xi^s_{(v_i,\ldots,v_k)}(t_{i+1}) - \frac{\kappa}{2} \\
&\overset{(5.12)}{\geq} \Psi^s(t_0) + \frac{(k-(i+1))\kappa}{2} - (1+\rho)(t_0 - t_{i+1}) \\
&\quad + L_{v_k}(t_0) - L_{v_k}(t_{i+1}). 
\end{aligned} \tag{5.14}$$

We need to show that $i+1 \neq k$ as claimed. Assuming the contrary, Inequality (5.14) leads to the contradiction

$$\begin{aligned}
0 = \xi^s_{(v_k)}(t_k) &\overset{(5.14)}{>} \Psi^s(t_0) - (1+\rho)(t_0 - t_k) + L_{v_k}(t_0) - L_{v_k}(t_k)\\
&\geq \Psi^s(t_0) - 2\rho(t_0 - t_k)\\
&\overset{(5.9)}{\geq} \Psi^s(t_0) - 2\rho\nabla^s\\
&\overset{(5.8)}{>} 0.
\end{aligned}$$

Hence it follows that indeed $i+1 < k$.

Recall that $t_{i+1} > t_0 - \nabla^s$ because $i \neq l$ and $i+1 < k$, i.e., time $t_{i+2}$ is defined. We obtain

$$\begin{aligned}
\xi^s_{(v_{i+1},\dots,v_k)}(t_{i+2}) &\geq \xi^s_{(v_{i+1},\dots,v_k)}(t_{i+1}) - (L_{v_{i+1}}(t_{i+1}) - L_{v_{i+1}}(t_{i+2}))\\
&\quad + L_{v_k}(t_{i+1}) - L_{v_k}(t_{i+2})\\
&\overset{(5.9)}{\geq} \xi^s_{(v_{i+1},\dots,v_k)}(t_{i+1}) - (1+\rho)(t_{i+1} - t_{i+2})\\
&\quad + L_{v_k}(t_{i+1}) - L_{v_k}(t_{i+2})\\
&\overset{(5.14)}{\geq} \Psi^s(t_0) + \frac{(k-(i+1))\kappa}{2} - (1+\rho)(t_0 - t_{i+2})\\
&\quad + L_{v_k}(t_0) - L_{v_k}(t_{i+2}),
\end{aligned}$$

i.e., the induction step succeeds.

**Part II. Getting further away from $v_k$.** We define a finite chain of nodes $w_l, \dots, w_m$ and times $t_{l+1} \geq t_{l+2} \geq \dots \geq t_{m+1} = t_0 - \nabla^s$, where $w_l := v_l$ and $t_{l+1}$ is the time at which the previous construction left off. The construction is inductive and maintains that for all $i \in \{l, \dots, m\}$ it holds that

$$\Xi^s_{w_i}(t_{i+1}) \geq \Psi^s(t_0) + ((1-\rho)\mu - 2\rho)(t_0 - t_{i+1}) \tag{5.15}$$

and also that

$$t_{i+1} = \min\{t \in [t_0 - \nabla^s, t_i] \,|\, L_{w_i}(t_i) - L_{w_i}(t) \leq (1+\rho)(t_i - t)\}, \tag{5.16}$$

First, we anchor an induction at index $l$. Observe that Equality (5.16) is satiesfied as it coincedes with Definition (5.9) for the indes $i = l$. Evaluating (5.12) for this index, we obtain for any $t \in [t_{l+1}, t_0]$ that

$$\begin{aligned}
\xi^s_{(v_l,\ldots,v_k)}(t) &= \xi^s_{(v_l,\ldots,v_k)}(t_{l+1}) + L_{v_l}(t) - L_{v_l}(t_{l+1}) \\
&\quad -(L_{v_k}(t) - L_{v_k}(t_{l+1})) \\
&\overset{(5.12)}{\geq} \Psi^s(t_0) - (1+\rho)(t_0 - t_{l+1}) + L_{v_l}(t) - L_{v_l}(t_{l+1}) \\
&\quad +L_{v_k}(t_0) - L_{v_k}(t) \qquad (5.17) \\
&\geq \Psi^s(t_0) - 2\rho(t_0 - t_{l+1}) \\
&\overset{(5.9)}{\geq} \Psi^s(t_0) - 2\rho\nabla^s \\
&\overset{(5.8)}{>} 0.
\end{aligned}$$

Note that $p \in P^s_{v_0}(t)$ for all $t \in [\underline{t}, t_0]$, as no edges are added to $E^s$ during this time interval due to the precondition that $T_s^{(r)} \notin \left(\underline{t}, \bar{t}\right]$ for all $r \in \mathbb{N}$. Thus, as $E_{(v_l,\ldots,v_k)} \subseteq E_p$, for all $t \in [\underline{t}, t_0]$, we have $\Xi^s_{v_l}(t) \geq \xi^s_{(v_l,\ldots,v_k)}(t) > 0$ during $[t_{l+1}, t_0] \subseteq \left[\underline{t}, \bar{t}\right]$. Applying Lemma 5.21 and Inequality (5.17) for time $t_0$, we see that

$$\begin{aligned}
\Xi^s_{v_l}(t_{l+1}) &\geq \Xi^s_{v_l}(t_0) - (L_{v_l}(t_0) - L_{v_l}(t_{l+1})) \qquad (5.18) \\
&\quad +(1+\mu)(1-\rho)(t_0 - t_{l+1}) \\
&\geq \xi^s_{(v_l,\ldots,v_k)}(t_0) - (L_{v_l}(t_0) - L_{v_l}(t_{l+1})) \\
&\quad +(1+\mu)(1-\rho)(t_0 - t_{l+1}) \\
&\overset{(5.17)}{\geq} \Psi^s(t_0) + ((1-\rho)\mu - 2\rho)(t_0 - t_{l+1}). \qquad (5.19)
\end{aligned}$$

Next, suppose the claim holds for index $i \in \{l, \ldots, m-1\}$. Thus, as $i \neq m$, $t_{i+1} > t_0 - \nabla^s$. If $i = l$, the previous construction must have halted because Inequality (5.10) was satisfied for some node $w \in \mathcal{N}^s_{v_l}(t_{l+1})$. In this case, we define $w_{l+1} := w$ such that

$$L_{w_{l+1}}(t_{l+1}) - L_{v_l}(t_{l+1}) > \left(s + \frac{1}{2}\right)\kappa.$$

On the other hand, if $i > l$, we claim that there also must exist a node $w_{i+1} \in \mathcal{N}^s_{w_i}(t_{i+1})$ fulfilling

$$L_{w_{i+1}}(t_{i+1}) - L_{w_i}(t_{i+1}) > \left(s + \frac{1}{2}\right)\kappa. \qquad (5.20)$$

Assuming for contradiction that this is not true, from the definition of $t_{i+1}$ and Lemma 5.23 we get that

$$L_{w_i}(t_{i+1}) - L_{w_{i-1}}(t_{i+1}) < \left(s + \frac{1}{2}\right)\kappa. \qquad (5.21)$$

Note that because $i > l$, we already have established Inequality (5.20) for index $i-1$. We infer that

$$\begin{aligned} L_{w_{i-1}}(t_i) - L_{w_{i-1}}(t_{i+1}) &\overset{(5.20,5.21)}{<} \left(L_{w_i}(t_i) - \left(s+\frac{1}{2}\right)\kappa\right) \\ &\quad - \left(L_{w_i}(t_{i+1}) - \left(s+\frac{1}{2}\right)\kappa\right) \\ &= L_{w_i}(t_i) - L_{w_i}(t_{i+1}) \\ &\overset{(5.16)}{\leq} (1+\rho)(t_i - t_{i+1}), \end{aligned}$$

which yields

$$\begin{aligned} L_{w_{i-1}}(t_{i-1}) - L_{w_{i-1}}(t_{i+1}) &= L_{w_{i-1}}(t_{i-1}) - L_{w_{i-1}}(t_i) \\ &\quad + L_{w_{i-1}}(t_i) - L_{w_{i-1}}(t_{i+1}) \\ &\overset{(5.16)}{<} (1+\rho)(t_{i-1} - t_{i+1}). \end{aligned}$$

This contradicts Equality (5.16), implying that indeed Statement (5.20) must hold true for an appropriate choice of node $w_{i+1}$.

Hence, whatever path maximizes $\Xi^s_{w_i}(t_{i+1})$, we can extend it by the edge $\{w_{i+1}, w_i\}$ in order to see that

$$\begin{aligned} \Xi^s_{w_{i+1}}(t_{i+1}) &\geq \Xi^s_{w_i}(t_{i+1}) + L_{w_{i+1}}(t_{i+1}) - L_{w_i}(t_{i+1}) - s\kappa \\ &\overset{(5.20)}{>} \Xi^s_{w_i}(t_{i+1}) \\ &\overset{(5.15)}{\geq} \Psi^s(t_0) + ((1-\rho)\mu - 2\rho)(t_0 - t_{i+1}). \end{aligned} \tag{5.22}$$

We now can define

$$t_{i+2} := \min\{t \in [t_0 - \nabla^s, t_{i+1}] \mid L_{w_{i+1}}(t_{i+1}) - L_{w_{i+1}}(t) \leq (1+\rho)(t_{i+1} - t)\}$$

in accordance with Equality (5.16). Since we have that

$$\Xi^s_{w_{i+1}}(t_{i+1}) \overset{(5.22)}{\geq} \Psi^s(t_0) \overset{(5.8,5.16)}{>} 2\rho(t_{i+1} - t_{i+2}),$$

Lemma 5.22 shows that $\Xi^s_{w_{i+1}}(t) > 0$ for all $t \in [t_{i+2}, t_{i+1}]$. Thus, applying Lemma 5.21 yields that

$$\begin{aligned} \Xi^s_{w_{i+1}}(t_{i+2}) &\geq \Xi^s_{w_{i+1}}(t_{i+1}) - (L_{w_{i+1}}(t_{i+1}) - L_{w_{i+1}}(t_{i+2})) \\ &\quad + (1+\mu)(1-\rho)(t_{i+1} - t_{i+2}) \\ &\overset{(5.16)}{\geq} \Xi^s_{w_{i+1}}(t_{i+1}) + ((1-\rho)\mu - 2\rho)(t_{i+1} - t_{i+2}) \\ &\overset{(5.15)}{\geq} \Psi^s(t_0) + ((1-\rho)\mu - 2\rho)(t_0 - t_{i+2}), \end{aligned}$$

i.e., the induction step succeeds. Note that the construction must halt after a finite number of steps because clock rates are bounded, we consider a finite time interval, and clock values increase by at least $(s+1/2)\kappa > 0$ whenever we move from a node $w_i$ to a node $w_{i+1}$. Thus, the induction is complete.

Inserting $i = m$ and $t_{m+1} = t_0 - \nabla^s$ into Inequality (5.15), we conclude that

$$\begin{aligned}\Xi^s_{w_m}(t_{m+1}) &\overset{(5.15)}{\geq} \Psi^s(t_0) + ((1-\rho)\mu - 2\rho)\nabla^s\\ &\overset{(5.8)}{>} (1-\rho)\mu\nabla^s\\ &\overset{(5.7)}{=} \kappa C_{s-1}.\end{aligned}$$

Therefore, using that $P^s_{w_m}(t_{m-1}) \subseteq P^{s-1}(t_{m-1})$, for any path $p'$ maximizing $\Xi^s_{w_m}(t_{m+1})$ we get

$$\begin{aligned}\Psi^{s-1}(t_{m+1}) &\geq \psi^{s-1}_{p'}(t_{m+1})\\ &\geq \xi^s_{p'}(t_{m+1})\\ &= \Xi^s_{w_m}(t_{m+1})\\ &> \kappa C_{s-1}.\end{aligned}$$

As $t_0 - \nabla^s \in \left[\underline{t}, \bar{t}\right]$, this contradicts the precondition that the system is $(s-1)$-legal during this time interval, finishing the proof. □

From this relation between legality on the levels $s-1$ and $s$ we can derive the main theorem of this section. Essentially, the lemma guarantees that the system is legal with respect to a gradient sequence with $C_s = C_{s-1}/\sigma$ except for the level where we added edges most recently. The careful definition of the insertion times $T_s^{(r)}$ makes sure that the algorithm waits for sufficiently long before proceeding to the next level, i.e., the given level will have stabilized again prior to losing the factor $\sigma$ decrease of $\Psi^{s+1}$ compared to $\Psi^s$ on the next level. Therefore, we need to "skip" one level in the respective gradient sequence, which is does not affect the asymptotic bounds.

**Theorem 5.25.** *At all times $t \in \mathbb{R}^+_0$, the system is legal with respect to the gradient sequence*

$$C := \left(\frac{\mathcal{G}}{\kappa}, \frac{\mathcal{G}}{\kappa}, \frac{\mathcal{G}}{\sigma\kappa}, \frac{\mathcal{G}}{\sigma^2\kappa}, \frac{\mathcal{G}}{\sigma^3\kappa}, \ldots\right).$$

*Proof.* Define for $s \in \mathbb{N}$ the gradient sequence $C^s$ by

$$\forall s' \in \mathbb{N}_0 : \; C^s_{s'} := \begin{cases} \mathcal{G}/(\sigma^{s'}\kappa) & \text{if } s' < s\\ \mathcal{G}/(\sigma^{s'-1}\kappa) & \text{otherwise.}\end{cases}$$

We claim that for all $r, s \in \mathbb{N}$ the system is legal with respect to $C^s$ during the time interval $\left[T_s^{(r)}, T_{s+1}^{(r)}\right)$. Note that because by definition we have for all $s \in \mathbb{N}$ and $s' \in \mathbb{N}_0$ that $C_{s'}^s \leq C_{s'}$, the statement of the theorem immediately follows as soon as this claim is established.

Assume for contradiction that the claim is false. Suppose that, for some $r, s \in \mathbb{N}$, $t_{\max} \in \left[T_s^{(r)}, T_{s+1}^{(r)}\right)$ is the infimum of all times when it is violated and $\bar{s} \in \mathbb{N}$ is the minimal level for which legality is violated at this time (recall that Statement $(i)$ of Lemma 5.20 gives that the system is 0-legal at all times), i.e.,

$$\Psi^{\bar{s}}\left(\tilde{t}_{\max}\right) > \kappa C_{\bar{s}}^s \tag{5.23}$$

for some time $\tilde{t}_{\max} > t_{\max}$ that is arbitrarily close to $t_{\max}$.

We make a case distinction. The first case is that $\bar{s} = s$. Since we have $C_s^s = C_{s-1}^s$ and w.l.o.g. the system is $(s-1)$-legal until time $\tilde{t}_{\max}$ (as $\bar{s}$ is minimal and $\tilde{t}_{\max} - t_{\max}$ is arbitrarily small), this is an immediate contradiction to Lemma 5.20, which states that the system is $s$-legal throughout the time interval $\left[T_s^{(r)}, \tilde{t}_{\max}\right)$.

The second case is that $\bar{s} \neq s$ and

$$t_{\max} < \frac{\mathcal{G}}{(1-\rho)\mu\sigma^{\min\{\bar{s}-2,0\}}},$$

implying that $r = s = 1$ and thus $\bar{s} \geq 2$. However, as $\mathcal{A}_\mu$ satisfies Definition 5.14 and no edges are added on level $\bar{s}$ until that time, we have that

$$\Psi^{\bar{s}}\left(t_{\max}\right) \leq 2\rho t_{\max} < \frac{2\rho\mathcal{G}}{(1-\rho)\mu\sigma^{\bar{s}-2}} \overset{(5.3)}{=} \frac{\mathcal{G}}{\sigma^{\bar{s}-1}} = \kappa C_{\bar{s}}^1.$$

This is a contradiction, as logical clocks are continuous and no edges are added to $E^{\bar{s}}$ during $(t_{\max}, \tilde{t}_{\max}] \subset \left(T_s^{(r)}, T_{s+1}^{(r)}\right)$, implying that $\Psi^{\bar{s}}$ does not increase at discontinuities during $(t_{\max}, \tilde{t}_{\max}]$.

The third case is that we have $\bar{s} > s$ and the second case does not apply. We claim that the system is $(\bar{s}-1)$-legal with respect to $C^s$ during the time interval

$$\left[t_{\max} - \frac{\kappa C_{\bar{s}-1}^s}{(1-\rho)\mu}, \tilde{t}_{\max}\right] \subseteq \left[t_{\max} - \frac{\mathcal{G}}{(1-\rho)\mu\sigma^{\bar{s}-2}}, \tilde{t}_{\max}\right] \subset \left[T_{\bar{s}}^{(r-1)}, T_{\bar{s}}^{(r)}\right),$$

where $T_{\bar{s}-1}^{(0)}$ is simply to be read as 0. This follows from the observations that $(i)$ $t_{\max}$ is sufficiently large for the left boundary to be at least 0 because Case 2 does not apply, $(ii)$ for $r \geq 2$ we have that

$$t_{\max} - \frac{\mathcal{G}}{(1-\rho)\mu\sigma^{\bar{s}-2}} \geq T_1^{(r)} - \frac{\mathcal{G}}{(1-\rho)\mu} \overset{(5.2,5.4)}{\geq} T_{\bar{s}-1}^{(r-1)},$$

(*iii*) the minimality of $\bar{s}$ ensures that the system must be $(\bar{s}-1)$-legal until some time that is strictly larger than $t_{\max}$, and (*iv*) the time difference $\tilde{t}_{\max} - t_{\max}$ can be chosen arbitrarily small. Therefore, we may apply Lemma 5.24 to the time $\tilde{t}_{\max}$ on level $\bar{s}$, yielding the contradiction

$$\Psi^{\bar{s}}(\tilde{t}_{\max}) \leq \frac{\kappa C^s_{\bar{s}-1}}{\sigma} = \kappa C^s_{\bar{s}} \tag{5.24}$$

to Inequality (5.23).

The fourth and final case is that $\bar{s} < s$. We get that the system is $(\bar{s}-1)$-legal with respect to $C^s$ during the interval

$$\left[t_{\max} - \frac{\kappa C^s_{\bar{s}-1}}{(1-\rho)\mu}, \tilde{t}_{\max}\right] \subseteq \left[T^{(r)}_{\bar{s}+1} - \frac{\mathcal{G}}{(1-\rho)\mu\sigma^{\bar{s}-1}}, \tilde{t}_{\max}\right] \overset{(5.2)}{\subset} \left[T^{(r)}_{\bar{s}}, T^{(r+1)}_{\bar{s}}\right),$$

which by Lemma 5.24 again implies the contradictory Inequality (5.24).

Since all possibilities lead to a contradiction, our initial assumption that $t_{\max}$ is finite must be wrong, concluding the proof. □

This theorem can be rephrased in terms of the gradient property and stabilization time of $\mathcal{A}_\mu$.

**Corollary 5.26.** *Algorithm $\mathcal{A}_\mu$ exhibits a stable gradient skew of*

$$\mathcal{S}_\mu(d) := \kappa d\left(\left\lceil \log_\sigma\left(\frac{\mathcal{G}}{\kappa d}\right)\right\rceil + \frac{5}{2}\right)$$

*with stabilization time*

$$T_\mu = \frac{2(2\sigma-1)\mathcal{G}}{(\sigma-1)(1-\rho)\mu}.$$

*Granted that $\mu \geq (2+\varepsilon)\rho/(1-\rho)$ for any constant $\varepsilon > 0$, $\kappa \in \mathcal{O}(U)$, and $\mathcal{G} \in \mathcal{O}(DU)$, we have*

$$\mathcal{S}_\mu \in \mathcal{O}\left(Ud \log_{\mu/\rho}\left(\frac{D}{d}\right)\right)$$

*and*

$$T_\mu \in \mathcal{O}\left(\frac{\mathcal{G}}{\mu}\right).$$

*Proof.* Since one round of $\mathcal{A}_\mu$ takes $T_\mu/2$ time, any edge that has been present for at least $T_\mu$ time existed during a complete round. Setting $s(d) := \lceil 1 + \log_\sigma(\mathcal{G}/(\kappa d))\rceil$, it is thus sufficient to show that at any time $t \in \mathbb{R}^+_0$ it holds for any path $p = (v, \ldots, w) \in P^{s(d)}(t)$ of length $\ell_p = d$ that

$$L_v(t) - L_w(t) \leq \left(s(d) + \frac{3}{2}\right)\kappa d.$$

Observe that we defined $s(d)$ such that $C_{s(d)} \leq d$ for the gradient sequence $C$ from Theorem 5.25. By the theorem and the definition of $s(d)$-legality we have that

$$\begin{aligned} L_v(t) - L_w(t) - \left(s(d) + \frac{1}{2}\right)\kappa d &= \psi_p^{s(d)}(t) \\ &\leq \Psi^{s(d)}(t) \\ &\leq \kappa C_{s(d)} \\ &\leq \kappa d, \end{aligned}$$

which can be rearranged to the desired inequality. □

We will conclude this section with exemplarily demonstrating two more interesting properties of $\mathcal{A}_\mu$. The first one is that the algorithm is self-stabilizing (cf. [26]).

**Definition 5.27** (Self-Stabilization). *An algorithm is called* self-stabilizing, *if it converges from an arbitrary initial state to a correct state with respect to a certain specification, i.e., after finite time the system remains in a correct state. It is called $T$-self-stabilizing, if this takes at most $T$ time units.*

A self-stabilizing algorithm is capable of recovering from arbitrary transient faults, since as soon as errors cease the system will restore a correct configuration. In our context, a correct state means that the definitions of $\mathcal{A}_\mu$ having a stable gradient skew of $\mathcal{S}_\mu$ with stabilization time $T_\mu$ are satisfied.

**Corollary 5.28.** *Suppose $\mathcal{A}_\mu$ is initialized at time $0$ with arbitrary logical clock values, however still the global skew of $\mathcal{G}$ is guaranteed at all times. Then $\mathcal{A}_\mu$ self-stabilizes within $T_\mu/2$ time, i.e., at all times $t \geq T_1^{(2)}$ the system is legal with respect to the gradient sequence $C$ given in Theorem 5.25.*

*Proof.* We modify the proof of Theorem 5.25 in that for all $s \in \mathbb{N}$ and times $t \in \left[T_s^{(1)}, T_{s+1}^{(1)}\right)$, the system is legal with respect to the gradient sequence $\tilde{C}^s$ given by

$$\forall s' \in \mathbb{N}_0 : \; \tilde{C}^s_{s'} := \begin{cases} \mathcal{G}/(\sigma^{s'}\kappa) & \text{if } s' < s \\ \mathcal{G}/(\sigma^{s-1}\kappa) & \text{otherwise,} \end{cases}$$

i.e., we give no non-trivial guarantees for levels $s' \geq s$ during the first round. Therefore, for $r = 1$ the case that $\bar{s} \geq s$ is covered by Statement (*ii*) from Lemma 5.20, i.e., all but the fourth case from the case differentiation in the proof become trivial. This case can be treated analogously to the theorem, as $C^s_{s'}$ and $\tilde{C}^s_{s'}$ coincide for all $s' \leq s$. Finally, the case that $r \geq 2$ is treated analogously as well. □

We remark that the sets $\mathcal{N}_v^s$, $v \in V$, $s \in \mathbb{N}$, are also part of the system's state. However, these sets are updated during each round of the algorithm and thus also stabilize correctly. Moreover, note that the technique to maintain a small global skew presented in Section 5.3 is also self-stabilizing, however with unbounded stabilization time, as the initial clock skew might be arbitrarily large. This drawback can be overcome by adding a mechanism to detect a (significant) violation of the bound on the global skew, overall resulting in an $\mathcal{O}(T_\mu)$-self-stabilizing algorithm (granted that clock estimates are also $\mathcal{O}(T_\mu)$-self-stabilizing).

An interesting side effect of this property is that, regardless of $\mathcal{G}$, a smaller maximum skew during a time interval of length at least $T_\mu$ will temporarily result in a smaller stable gradient skew. However, without further modification, the stabilization time of the algorithm with respect to integrating new edges remains $T_\mu$, as the algorithm is not aware of the fact that edges could be incorporated more quickly.

The second property of $\mathcal{A}_\mu$ we would like to highlight is that, by slight abuse of notation, the algorithm by itself ensures a global skew linear in $U$ and the "classical" diameter of $G_{T_\mu}$.

**Corollary 5.29.** *For all $t \in \mathbb{R}_0^+$, denote by $D(t)$ the diameter of $G_{T_\mu}(t)$ and assume that $D(t) \leq D_{\max}$ for all $t$. If $\mu \geq (1+\varepsilon)4\rho/(1-\rho)$ for some constant $\varepsilon > 0$ and $\kappa \in \mathcal{O}(U)$, we have a global skew of*

$$\mathcal{G}_\nabla := \frac{3(1-\rho)\mu}{2((1-\rho)\mu - 4\rho)}\kappa D_{\max} \in \mathcal{O}(U D_{\max}).$$

*Proof.* Assume w.l.o.g. that $\mathcal{G}_\nabla \leq \mathcal{G}$. Suppose for the sake of contradiction that $t_{\max} \in \mathbb{R}_0^+$ is the infimum of all times when the global skew of $\mathcal{G}_\nabla$ is violated, i.e., for all $t \leq t_{\max}$ we have that $\max_{v,w\in V}\{L_v(t) - L_w(t)\} \leq \mathcal{G}_\nabla$. Define

$$\tilde{\nabla}^1 := \frac{\mathcal{G}_\nabla}{(1-\rho)\mu}.$$

Trivially, we have for any node $v \in V$ that $\Xi_v^1(t) < \mathcal{G}_\nabla$ at all times $t \leq t_{\max}$. Reasoning as in Lemma 5.24 for $s = 1$, but with $\nabla^1$ replaced by $\tilde{\nabla}^1$, we see that for all $r \in \mathbb{N}$ and $t \leq t_{\max}$ we have

$$t \in \left[T_1^{(r)} + \tilde{\nabla}^1, T_1^{(r+1)}\right) \Rightarrow \Psi^1(t) \leq 2\rho\tilde{\nabla}^1.$$

Thus, at such times $t$, we have for any two nodes $v, w \in V$ joined by a shortest path $p \in P_v^1(t)$ that

$$L_v(t) - L_w(t) \leq \Psi^1(t) + \frac{3}{2}\kappa\ell_p \leq 2\rho\tilde{\nabla}^1 + \frac{3}{2}\kappa D_{\max} = \mathcal{G}_\nabla - 2\rho\tilde{\nabla}^1.$$

Therefore, $t_{\max} \notin \left[T_1^{(r)} + \tilde{\nabla}^1, T_1^{(r+1)}\right)$ for any $r \in \mathbb{N}$. Moreover, since clock values are continuous and $\tilde{\nabla}^1 \leq \nabla^1 < T_\mu/2$ because $\mathcal{G}_\nabla \leq \mathcal{G}$, this implies for any $r \in \mathbb{N}$ with $T_1^{(r)} \leq t_{\max}$ that

$$\max_{v,w \in V} \left\{L_v\left(T_1^{(r)}\right) - L_w\left(T_1^{(r)}\right)\right\} \leq \mathcal{G}_\nabla - 2\rho\tilde{\nabla}^1.$$

Considering that $\max_{v,w\in V}\{L_v(t) - L_w(t)\}$ grows at most at rate $2\rho$ since $\mathcal{A}_\mu$ satisfies Definition 5.14, we see that also $t_{\max} \notin \left[T_1^{(r)}, T_1^{(r)} + \tilde{\nabla}^1\right]$ for any $r$ with $T_1^{(r)} \leq t_{\max}$.

Now let $r_{\max} \in \mathbb{N}$ be maximal such that $T_1^{(r_{\max})} \leq t_{\max}$. Combining the previous observations, we conclude that $t_{\max} \notin \left[T_1^{(r_{\max})}, T_1^{(r_{\max}+1)}\right]$, contradicting the definition of $r_{\max}$. □

We remark that since $\mathcal{A}_\mu$ is also self-stabilizing, indeed it is sufficient that $D(t) \leq D_{\max}$ for a sufficient period of time in the recent past in order to ensure the stated bound. However, employing a flooding mechanism to ensure a small global skew is more reliable, as it tolerates a much weaker connectivity of the dynamic graph $G$.

## 5.6 Discussion

Throughout this chapter, we employed a greatly simplified model in order to facilitate the presentation of $\mathcal{A}_\mu$. In this section, we will discuss our assumptions and briefly sketch some existing and possible generalizations. Note that all proposed adaptions of $\mathcal{A}_\mu$ are mutually compatible, i.e., all addressed issues may be dealt with simultaneously.

**Bounded local memory, computation, and number of events in finite time.** The way we described $\mathcal{A}_\mu$, in each round $r \in \mathbb{N}$ there are infinitely many times $T_s^{(r)}$ when each node $v \in V$ needs to perform state changes, infinitely many sets $\mathcal{N}_v^s$ to store, etc. However, as we have finite skew bounds, for each node there is a maximal level $s_v \in \mathbb{N}$ up to which these times and sets are relevant. This value can be bounded by

$$s_v \leq \left\lceil 1 + \log_\sigma\left(\frac{\mathcal{G}}{\kappa}\right)\right\rceil$$

and $v$ needs to check the algorithm's conditions only up to that level.

**Asynchronous edge insertions.** It is mandatory to drop the assumption that all nodes synchronously insert new edges on level $s$ at the times $T_s^{(r)}$, $r \in \mathbb{N}$. This can be elegantly solved by adding edges at certain logical clock values instead of fixed points in time. Adding an additional "buffer time" ensures that level $s$ stabilized at least *locally* before advancing to level $s+1$. Putting it simply, the skew bounds from level $s-1$ are tight enough to synchronize edge insertions on level $s$ sufficiently precisely to maintain a bound of $\Theta(\mathcal{G}/\mu)$ on the overall stabilization time. More specifically, the key observations are that $(i)$ we stabilize new edges inductively starting from low levels, $(ii)$ at the time when we insert a new edge adjacent to a node $v$, up to certain distances from $v$ the lower levels $s' < s$ will already have stabilized, $(iii)$ the skew bounds from these levels $s' < s$ make sure that nodes that are close to $v$ will not insert their newly appeared adjacent edges much earlier or later than $v$ on level $s$, and $(iv)$ paths that are longer than $C_{s'}$ for some $s' < s$ will not maximize $\Psi^s$, because $s'$-legality ensures that they exhibit an average skew smaller than $(s+1/2)\kappa$, i.e., we do not need to care about distant nodes. Roughly speaking, instead of reasoning globally about $\Psi^s$ and the time intervals $\left[T_s^{(r)}, T_{s+1}^{(r)}\right)$, we argue locally about some properly defined $\Psi_v^s$ and time intervals $\left[L_v^{-1}\left(T_s^{(r)}\right), L_v^{-1}\left(T_{s+1}^{(r)}\right)\right)$ measured in terms of the logical time of $v$. Naturally, localizing the proofs and relating local times of different nodes adds a number of technical difficulties. We refer the interested reader to [51] for details.

**Asynchronous edge arrivals and departures.** Another unrealistic assumption we made is that both endpoints of an edge detect its appearance or disappearance at exactly the same point in time. Obviously, this is not possible in practice, as nodes are not perfectly synchronized, send only finitely many messages, etc. If there is some delay $\tau$ between the two nodes recognizing the event, one of them will still act as if the edge was still operational for up to $\tau$ time while the other does not. It is not hard to believe that, since nodes can estimate each other's progress during $\tau$ time up to $\mathcal{O}(\mu\tau)$ by means of their hardware clock, increasing $\kappa$ by $\mathcal{O}(\mu\tau)$ is enough to address this problem. Although this is true, e.g. the fact that $\Psi_v^s$ and $\Xi_v^s$ cannot be related to nodes' actions as cleanly and directly as before increases the complexity of the proofs considerably. Again, we point to [51] for a full proof respecting these issues.

**Non-differentiable, discrete, and/or finite clocks.** We assumed that hardware and logical clocks are differentiable in order to support the reader's intuition. The proofs however rely on the progress bounds of the clocks only. In a practical system, hardware and logical clocks will both have finite and discrete values. Putting it simply, having discrete clocks does not change anything except that the clock resolution imposes a lower bound on the uncertainty $U$ (see also [59]). The size of clock values can be kept finite by standard wrap-around techniques, where one needs to make sure that the range of the clocks exceeds $2\mathcal{G}$ in order to be able to compare values correctly at all times.

**Initialization.** Initialization can be done by flooding. For many reasonable sets of system parameters, the initial clock skews will then be small enough not to violate the gradient skew constraints (cf. [59]). Alternatively, one can simply treat all edges as newly appeared and wait for the algorithm to stabilize. Note that a node that arrives late can set its clock value to the first one it becomes aware of without violating the global skew constraint. However, a (long-lasting) partition of the system necessitates to fall back to self-stabilizing solutions, temporarily jeopardizing the gradient property for all but one of the previous components.

**Non-uniform uncertainties.** In almost any practical setting, links will be different. Consequently, we would like $\mathcal{A}_\mu$ to deal with different uncertainties $U_e$ for each edge $e \in E(t)$. This is easily achieved by applying the same rules as $\mathcal{A}_\mu$ with $\kappa$ replaced by $\kappa_e > 2U_e$ (respectively $\kappa \in \Omega(U_e + \mu\tau_e)$, where $\tau_e$ replaces the value $\tau$ from above). In contrast to the previous issues, this does not affect the analysis at all—essentially we replace $\kappa$ by $\kappa_e$ also there (see [50, 51, 52, 55]). In general, uncertainties may also be a function of time, e.g. due to fluctuations in wireless link quality. This can be addressed, too, as we will see in a moment.

**Short-lived edge failures.** If an edge fails for a short period of time, it is not desirable that we have to wait for $\Omega(T_\mu)$ time until again a non-trivial gradient property holds. Also here one could make use of the fact that the estimates nodes have of their neighbors deteriorate at rate at most $\mathcal{O}(\mu)$ in absence of communication. The idea is to gradually increase the uncertainty of a failed edge accordingly and decrease it again once the edge returns to being operational. This has to happen slowly, though, leading to a suboptimal stabilization time when applied to an edge that has been absent for a long period of time or is completely new [50]. However, we conjecture that this technique can be freely combined with the edge integration scheme of $\mathcal{A}_\mu$ in order to achieve fast stabilization in both cases. Moreover, it offers a seamless solution to the aforementioned problem of edge uncertainties that vary over time.

**Directed Graphs.** Our model considers simple graphs only, i.e., there are no unidirectional links and uncertainties are symmetric. This requirement is crucial for $\mathcal{A}_\mu$ to work. However, we made no assumption on how nodes obtain clock estimates of their neighbors. To give but one example on how to derive a symmetric estimate graph from an asymmetric communication infrastructure, consider the following setting. Assume that estimates are computed from direct communication and we have a strongly connected directed graph. For any two neighbors, the receiving node may respond to each message with the estimated clock difference via multihop communication. The returned estimate will lose accuracy in the order of $\mu\tau$ if it travels for $\tau$ time. As long as $\mu\tau$ is not too large (note that choosing $\mu := \sqrt{\rho}$ still yields an asymptotically optimal gradient property), this way accurate estimates can be obtained. The value of $U_e$ for link $e \in \binom{V}{2}$ then simply is the larger of the two accuracy bounds.

**Unrestrained clock rates.** We imposed that $\mu \in \mathcal{O}(1)$. A lower bound from [60] shows that permitting larger clock rates is of no use when trying to achieve a stronger gradient property. One might now ask why we cannot obtain skew bounds that are linear in the distance between each pair of nodes by making $\sigma \in \Theta(\mu/\rho)$ sufficiently large such that the logarithm in $\mathcal{S}_\mu$ is bounded by a constant. The problem that arises is that if the algorithm wants to make use of a large value of $\mu \in \omega(1)$, this entails that also the uncertainty $U$ must grow linearly in $\mu$, as there is insufficient time to communicate the intended clock progress first. However, as mentioned in Section 5.2, a larger value of $\mu$ might help to achieve a better stabilization time if one relaxes the gradient property.

Opposed to that, the main impact of permitting clock rates of $o(1)$ indeed is that a skew bound linear in the distance can be maintained. Phrasing it differently, one can pose the question what is the strongest possible guarantee on the progress of logical clocks if we require a skew of $\mathcal{O}(U)$ between neighbors. A simple transformation answers this question by means of Theorem 5.25 and a second lower bound from [60]. If we slow down logical clock rates by factor $f := \log_\sigma(\mathcal{G}/(UD))$ as soon as the clock skew to any neighbor exceeds $\Omega(U)$, nodes are capable of determining logical clock skews to neighbors up to $\mathcal{O}(U/f)$ whenever they exceed some threshold $\Omega(U)$. Thus, we can "scale down" the fast and slow conditions on higher levels, i.e., substitute $\kappa$ by some value in $\mathcal{O}(\kappa/f)$ and still guarantee that the conditions can be implemented. Therefore, analogously to Corollary 5.26, we will get a skew bound of $\mathcal{O}(Ud)$ for paths of length $d$. The lower bound from [60] tells that this bound is asymptotically optimal for clock rates of $\Omega(1/f)$.

In some sense, this can be read as a statement about synchronizers: A synchronizer needs to guarantee a skew of $\mathcal{O}(U)$ between neighbors in order to locally trigger a synchronous round every $\mathcal{O}(U)$ logical time. Thus, the derived bounds state that under this constraint we can guarantee that any node can trigger a round locally at least every $\mathcal{O}(fU)$ real time, but not better. In contrast, a traditional synchronizer might stall a node for up to $\Omega(D)$ time, where $D$ denotes the diameter of the communication graph in question.

**Unknown system parameters.** As can be seen from the analysis, asymptotically optimal global skew, stable gradient skew, and stabilization time can be achieved without knowledge on the drift $\rho$. In fact, we merely need $\rho$ to be bounded away from one in order to compute sufficiently large values $\mu$, $T_\mu$, etc. Knowledge on $\rho$ is required only if we want to determine the minimal feasible value of $\mu$ due to the condition that $\mu > 2\rho/(1-\rho)$.

In contrast, it is crucial to have a reasonable upper bound on the uncertainty $U$, as it affects the gradient property almost linearly through $\kappa$. Similarly, we need a good bound on $\tau$ if we consider the setting where nodes detect edge arrivals and departures asynchronously. We remark that in practice it is a viable option to employ optimistic bounds on $U$ and $\tau$ and adapt them if they turn out to be too small. The fact that the executions leading to large skews require "everything to go wrong" for a considerable amount of time together with the strong stabilization properties of $\mathcal{A}_\mu$ indicate that such a scheme should be quite robust to all kinds of dynamics.

**Adaptive global skew.** The one "parameter", so to speak, that remains to be understood is the global skew $\mathcal{G}$. In fact, it is relevant for the stabilization time of the algorithm only, as the analysis of the gradient property can be conducted with the "true" global skew, i.e., $\max_{v,w\in V,t}\{L_v(t)-L_w(t)\}$, as long as the estimate $\mathcal{G}$ of this value the algorithm uses is a valid upper bound at all times. However, ideally the algorithm should be able to incorporate new edges quickly whenever the maximum skew is small. This is particularly important in highly dynamic settings, where on the one hand we strive for resilience to temporary weak connectivity (e.g. a list or even a disconnected graph), but also for a small stabilization time whenever the topology is more benign. To this end, a strong estimator is in demand that provides the nodes with an upper bound on the current maximum skew in the network. Based on such an estimator, nodes can change the duration of a round appropriately to achieve good stabilization times in an adaptive manner.

In a static graph, the diameter can be determined easily by a flooding-echo initiated by some leader. The determined value then is distributed by a second flooding to all nodes. In a dynamic graph, this approach becomes difficult as the dynamic diameter of the graph might increase while a flooding is in progress. However, if the diameter of the graph increases considerably, the global skew bound does also, i.e., it is feasible to spend a larger amount of time to incorporate new edges. Moreover, the actual maximal clock skew between any two nodes increases slowly at a rate of at most $2\rho$. Thus, it is possible to use an approach where only nodes $v \in V$ that receive an update on the estimate of the global skew participate in a round, that is, add new edges to their neighborhood subsets $\mathcal{N}_v^s$. The participating nodes can be certain that the used bound remains valid until their new edges have stabilized. The remaining nodes simply wait until a flooding-echo-flooding sequence for the new, larger diameter terminated (from their perspective) successfully before adding their edges in the respective round.

This simple approach achieves the desired goal, but suffers from a very large message size. In order to be certain that a flooding terminated, *all* nodes need to report back to the leader. More sophisticated techniques are in demand in order to obtain an algorithm that is both practical and adaptive with respect to the global skew.

## Part II

# Load Balancing

Chapter 6

# An Introduction to Parallel Randomized Load Balancing

*"Divide and conquer." – Gaius Julius Caesar.*

Apart from fault-tolerance, maybe *the* main motivation to study distributed computing is parallelism. If a task cannot be solved fast enough with a single processor, just use more! Nowadays it is common folklore that this approach does help to a certain degree, but eventually hits fundamental barriers. Sorting $n$ items, for instance, requires $\mathcal{O}((n \log n)/k)$ rounds on $k \leq n$ processors, whereas in general one cannot be faster than $\Omega(\log n)$ rounds – regardless of the amount of hardware thrown at the problem.

In this part of this thesis, we will examine a more subtle obstacle to parallelism in a distributed world. Even if the considered problem is "embarrassingly parallel", *coordinating* a distributed system's efforts to solve it often incurs an overhead. To achieve perfect scalability, also coordination must be parallel, i.e., one cannot process information sequentially, or collect the necessary coordination information at a single location. A striking and fundamental example of coordination is load balancing, which occurs on various levels: canonical examples are job assignment tasks such as sharing work load among multiple processors, servers, or storage locations, but the problem also plays a vital role in e.g. low-congestion circuit routing, channel bandwidth assignment, or hashing, cf. [86].

A common archetype of all these tasks is the well-known balls-into-bins problem: Given $n$ balls and $n$ bins, how can one place the balls into the bins quickly while keeping the maximal bin load small? As in other areas where centralized control must be avoided (sometimes because it is impossible), the key to success is *randomization*. Adler et al. [1] devised parallel algorithms for the problem whose running times and maximal bin loads are essentially doubly-logarithmic. They provide a lower bound which is asymptotically matching the upper bound. However, their lower bound proof requires two critical restrictions: algorithms must (*i*) break ties *symmetrically* and (*ii*) be *non-adaptive*, i.e., each ball restricts itself to a fixed number of candidate bins before communication starts. Dropping these assumptions, we are able to devise more efficient algorithms.

More precisely, we are going to present a simple adaptive and symmetric algorithm achieving a maximal bin load of two within $\log^* n + \mathcal{O}(1)$ rounds of communication w.h.p. This is achieved using an asymptotically optimal number of messages. Complementing this upper bound, we prove that—given the constraints on bin load and communication complexity—the running time of our first algorithm is $(1 + o(1))$-optimal for symmetric algorithms. Our bound necessitates a new proof technique; it is not a consequence of the impossibility to gather reliable information in time (e.g. due to asynchronicity, faults, or explicitly limited local views of the system), rather it emerges from bounding the total amount of communication. Thus, we demonstrate that breaking symmetry to a certain degree, i.e., reducing entropy far enough to guarantee small bin loads, comes at a cost exceeding the apparent minimum of $\Omega(n)$ total bits and $\Omega(1)$ rounds. In this light, a natural question to pose is how much initial entropy is required for the lower bound to hold. We show that the crux of the matter is that bins are initially anonymous, i.e., balls do not know globally unique addresses of the bins. This captures the essence of the aforementioned condition of symmetry imposed by Adler et al. Discarding this requirement, we give an asymmetric adaptive algorithm that runs in constant time and sends $\mathcal{O}(n)$ messages w.h.p., yet achieves a maximal bin load of three. Completing the picture, we show that the same is possible for a symmetric algorithm if we incur a slightly superlinear number of messages or non-constant maximal bin loads. Jointly, the given bounds provide a full characterization of the parallel complexity of the balls-into-bins problem.

## 6.1 Model

The system consists of $n$ bins and $n$ balls, and we assume it to be fault-free. We employ a synchronous message passing model, where one round consists of the following steps:

1. Balls perform (finite, but otherwise unrestricted) local computations and send messages to arbitrary bins.

2. Bins receive these messages, do local computations, and send messages to any balls they have been contacted by in this or earlier rounds.

3. Balls receive these messages and may commit to a bin (and terminate).

Note that (for reasonable algorithms) the third step does not interfere with the other two. Hence, the literature typically accounts for this step as "half a round" when stating the time complexity of balls-into-bins algorithms; we adopt this convention.

The considered task now can be stated concisely.

**Problem 6.1** (Parallel Balls-into-Bins)**.** *We want to place each ball into a bin. The goals are to minimize the total number of rounds until all balls are placed, the maximal number of balls placed into a bin, and the amount of involved communication.*

In order to classify balls-into-bins algorithms, we fix the notions of an algorithm being adaptive or symmetric, respectively.

**Definition 6.2** (Adaptivity)**.** *A balls-into-bins algorithm is* non-adaptive*, if balls fix the subset of the bins they will contact prior to all communication. An algorithm not obeying this constraint is called* adaptive*.*

A natural restriction for algorithms solving Problem 6.1 is to assume that random choices cannot be biased, i.e., also bins are anonymous. This is formalized by the following definition.

**Problem 6.3** (Symmetric Balls-into-Bins)**.** *We call an instance of Problem 6.1* symmetric*, if balls and bins identify each other by u.i.r. port numberings. We call an algorithm that can be implemented under this constraint* symmetric*.*

In contrast, balls executing an asymmetric algorithm may e.g. all decide to contact bin 42. Note that this is impossible for symmetric algorithms, for which the uniformly random port numberings even out any non-uniformity in the probability distribution of contacted port numbers.

**Problem 6.4** (Asymmetric Balls-into-Bins)**.** *An instance of Problem 6.1 is* asymmetric*, if balls identify bins by globally unique addresses* $1, \ldots, n$*. An algorithm relying on this information is called* asymmetric*.*

## 6.2 Related Work

Probably one of the earliest applications of randomized load balancing has been hashing. In this context, it was proved that when throwing $n$ balls u.i.r. into $n$ bins, the fullest bin has load $(1 + o(1)) \log n / \log \log n$ in expectation [41]. It is also common knowledge that the maximal bin load of this simple approach is $\Theta(\log n / \log \log n)$ w.h.p. (e.g. [29]).

With the growing interest in parallel computing, since the beginning of the nineties the topic received increasingly more attention. Karp et al. [44] demonstrated for the first time that two random choices are superior to one. By combining two (possibly not fully independent) hashing functions, they simulated a parallel random access machine (PRAM) on a distributed memory machine (DMM) with a factor $\mathcal{O}(\log\log n \log^* n)$ overhead; in essence, their result was a solution to balls-into-bins with a maximal bin load of $\mathcal{O}(\log\log n)$ w.h.p. Azar et al. [6] generalized their result by showing that if the balls choose sequentially from $d \geq 2$ u.i.r. bins greedily the currently least loaded one, the maximal load is $\log\log n/\log d + \mathcal{O}(1)$ w.h.p.[1] They prove that this bound is stochastically optimal in the sense that any other strategy to assign the balls majorizes[2] their approach. The expected number of bins each ball queries during the execution of the algorithm was later improved to $1+\varepsilon$ (for any constant $\varepsilon > 0$) by Czumaj and Stemann [22]. This is achieved by placing each ball immediately if the load of an inspected bin is not too large, rather then always querying $d$ bins.

So far the question remained open whether strong upper bounds can be achieved in a distributed setting. Adler et al. [1] answered this affirmatively by devising a parallel greedy algorithm obtaining a maximal load of $\mathcal{O}(d + \log\log n/\log d)$ within the same number of rounds w.h.p. Thus, choosing $d \in \Theta(\log\log n/\log\log\log n)$, the best possible maximal bin load of their algorithm is $\mathcal{O}(\log\log n/\log\log\log n)$. On the other hand, they prove that a certain subclass of algorithms cannot perform better with probability larger than $1 - 1/\operatorname{polylog} n$. The main characteristics of this subclass are that algorithms are *non-adaptive*, i.e., balls have to choose a fixed number of $d$ candidate bins before communication starts, and *symmetric*, i.e., these bins are chosen u.i.r. Moreover, communication takes place only between balls and their candidate bins. In this setting, Adler et al. show also that for any constant values of $d$ and the number of rounds $r$ the maximal bin load is in $\Omega((\log n/\log\log n)^{1/r})$ with constant probability. Recently, Even and Medina extended their bounds to a larger spectrum of algorithms by removing some artificial assumptions [32]. A matching algorithm was proposed by Stemann [100], which for $d = 2$ and $r \in \mathcal{O}(\log\log n)$ achieves a load of $\mathcal{O}((\log n/\log\log n)^{1/r})$ w.h.p.; for $r \in \Theta(\log\log n)$ this implies a constantly bounded bin load. The only “adaptive” algorithm proposed so far is due to

[1]There is no common agreement on the notion of w.h.p. Frequently it refers to probabilities of at least $1 - 1/n$ or $1 - o(1)$, as so in the work of Azar et al.; however, their proof also provides their result w.h.p. in the sense we use throughout this thesis.

[2]Roughly speaking, this means that any other algorithm is as least as likely to produce bad load vectors as the greedy algorithm. An $n$-dimensional load vector is worse than another, if after reordering the components of both vectors descendingly, any partial sum of the first $i \in \{1, \dots, n\}$ entries of the one vector is greater or equal to the corresponding partial sum of the other.

Even and Medina [31]. If balls cannot be allocated, they get an additional random choice. However, one could also give all balls this additional choice right from the start, i.e., this kind of adaptivity cannot circumvent the lower bound. Consequently, their 2.5 rounds algorithm uses a constant number of choices and exhibits a maximal bin load of $\Theta(\sqrt{\log n/\log\log n})$ w.h.p., the same asymptotic characteristics as parallel greedy with 2.5 rounds and two choices. In comparison, within this number of rounds our technique is capable of achieving bin loads of $(1 + o(1)) \log\log n/\log\log\log n$ w.h.p.[3] See Table 6.1 for a comparison of our results to parallel algorithms. Our adaptive algorithms outperform all previous solutions for the whole range of parameters.

Given the existing lower bounds, the only possibility for further improvement has been to search for non-adaptive or asymmetric algorithms. Vöcking [103] introduced the sequential "always-go-left" algorithm which employs asymmetric tie-breaking in order to improve the impact of the number of possible choices $d$ from logarithmic to linear. Furthermore, he proved that dependency of random choices does not offer asymptotically better bounds. His upper bound holds also true if merely two bins are chosen randomly, but for each choice $d/2$ consecutive bins are queried [45]. Table 6.2 summarizes sequential balls-into-bins algorithms. Note that not all parallel algorithms can also be run sequentially. Stemann's collision protocol, for instance, requires bins to accept balls only if a certain number of pending requests is not exceeded. Thus the protocol cannot place balls until all balls' random choices are communicated. In contrast, our approach translates to a simple sequential algorithm competing in performance with the best known results [22, 103]. This algorithm could be interpreted as a greedy algorithm with $d = \infty$.

Beyond that, there is a huge body of work studying variants of the basic problem [4, 6, 13, 14, 15, 22, 47, 48, 82, 83, 84, 85, 93, 100, 102, 103, 104]. In this exposition we will focus on the simple version of the task. A brief overview of these works can be found in [68].

Results related to ours have been discovered before for hashing problems. A number of publications presents algorithms with running times of $\mathcal{O}(\log^* n)$ (or very close) in PRAM models [10, 38, 78, 81]. At the heart of these routines as well as our balls-into-bins solutions lies the idea to use an in each iteration exponentially growing share of the available resources to deal with the remaining keys or bins, respectively. Implicitely, this approach already occured in previous work by Raman [94]. For a more detailed review of

[3]This follows by setting $a := (1 + \varepsilon) \log\log n/\log\log\log n$ (for arbitrary small $\varepsilon > 0$) in the proof of Corollary 8.6; we get that merely $n/(\log n)^{1+\varepsilon}$ balls remain after one round, which then can be delivered in 1.5 more rounds w.h.p. using $\mathcal{O}(\log n)$ requests per ball.

Table 6.1: Comparison of parallel balls-into-bins algorithms. Commiting balls into bins counts as half a round with regard to time complexity.

| algorithm | sym. | adpt. | choices | rounds | maximal bin load | messages |
|---|---|---|---|---|---|---|
| naive [41] | yes | no | 1 | 0.5 | $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ | $n$ |
| par. greedy [1] | yes | no | 2 | 2.5 | $\mathcal{O}\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ | $\mathcal{O}(n)$ |
| par. greedy [1] | yes | no | $\Theta\left(\frac{\log\log n}{\log\log\log n}\right)$ | $\Theta\left(\frac{\log\log n}{\log\log\log n}\right)$ | $\mathcal{O}\left(\frac{\log\log n}{\log\log\log n}\right)$ | $\mathcal{O}\left(\frac{n\log\log n}{\log\log\log n}\right)$ |
| collision [100] | yes | no | 2 | $r+0.5$ | $\mathcal{O}\left(\left(\frac{\log n}{\log\log n}\right)^{1/r}\right)$ | $\mathcal{O}(n)$ |
| $\mathcal{A}_b^2$ | yes | yes | $\mathcal{O}(1)$ (exp.) | $\log^* n+\mathcal{O}(1)$ | 2 | $\mathcal{O}(n)$ |
| $\mathcal{A}_b(r)$ | yes | yes | $\mathcal{O}(1)$ (exp.) | $r+\mathcal{O}(1)$ | $\frac{\log^{(r)} n}{\log^{(r+1)} n}+r+\mathcal{O}(1)$ | $\mathcal{O}(n)$ |
| $\mathcal{A}_c(l)$ | yes | yes | $\mathcal{O}(l)$ (exp.) | $\log^* n-\log^* l+\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(ln)$ |
| $\mathcal{A}(\sqrt{\log n})$ | no | yes | $\mathcal{O}(1)$ (exp.) | $\mathcal{O}(1)$ | 3 | $\mathcal{O}(n)$ |

Table 6.2: Comparison of sequential balls-into-bins algorithms.

| algorithm | sym. | adpt. | choices | max. bin load | bin queries |
|---|---|---|---|---|---|
| naive [41] | yes | no | 1 | $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ | $n$ |
| greedy [6] | yes | no | $d\geq 2$ | $\frac{\log\log n}{\log d}+\mathcal{O}(1)$ | $\mathcal{O}(dn)$ |
| always-go-left [103] | no | no | $d\geq 2$ | $\mathcal{O}\left(\frac{\log\log n}{d}\right)$ | $\mathcal{O}(dn)$ |
| adpt. greedy [22] | yes | yes | $1+o(1)$ (exp.); at most $d\geq 2$ | $\mathcal{O}\left(\frac{\log\log n}{\log d}\right)$ | $(1+o(1))n$ |
| $\mathcal{A}_{\text{seq}}$ | yes | yes | $\mathcal{O}(1)$ (exp.) | 2 | $(2+o(1))n$ |

these papers, we refer the interested reader to [42]. Despite differences in the models, our algorithms and proofs exhibit quite a few structural similarities to the ones applicable to hashing in PRAM models. From our point of view, there are two main differences distinguishing our upper bound results on symmetric algorithms. Firstly, the parallel balls-into-bins model permits to use the algorithmic idea in its most basic form. Hence, our presentation focuses on the properties decisive for the $\log^* n + \mathcal{O}(1)$ complexity bound of the basic symmetric algorithm. Secondly, our analysis shows that the core technique is highly robust and can therefore tolerate a large number of faults.

The lower bound by Adler et al. (and the generalization by Even and Medina) is stronger than our lower bound, but it applies to algorithms which are severely restricted in their abilities only. Essentially, these restrictions uncouple the algorithm's decisions from the communication pattern; in particular, communication is constrained to an initially fixed random graph, where each ball contributes $d$ edges to u.i.r. bins. This prerequisite seems reasonable for systems where the initial communication overhead is large. In general, we believe it to be difficult to motivate that a non-constant number of communication rounds is feasible, but an initially fixed set of bins may be contacted only. In contrast, our lower bound also holds for adaptive algorithms; in other words, it arises from the assumption that bins are (initially) anonymous, which fits a wide range of real-world systems.

Like Linial in his seminal work on 3-coloring the ring [70], we attain a lower bound of $\Omega(\log^* n)$ on the time required to solve the task efficiently. This connection is more than superficial, as both bounds essentially arise from a symmetry breaking problem. However, Linial's argument just uses a highly symmetric ring topology. This general approach to argue about a simple topology has been popular when proving lower bounds (see e.g. [25, 65, 89]). This is entirely different from our setting, where any two parties may potentially exchange information. Therefore, we cannot argue on the basis that nodes will learn about a specific subset of the global state contained within their local horizon only. Instead, the random decisions of a balls-into-bins algorithm define a graph describing the flow of information. This graph is not a simple random graph, as the information gained by this communication feeds back to its evolution over time, i.e., future communication may take the local topology of its current state into account.

A different lower bound technique is by Kuhn et al. [54], where a specific locally symmetric, but globally asymmetric graph is constructed to render a problem hard. Like in our work, [54] restricts its arguments to graphs which are locally trees. The structure of the graphs we consider imposes to examine subgraphs which are trees as well; subgraphs containing cycles occur too infrequently to constitute a lower bound. The bound of $\Omega(\log^* n)$ from [38], applicable to hashing in a certain model, which also argues about trees, has even more in common with our result. However, neither of these bounds needs to deal with the difficulty that the algorithm may influence the evolution of the communication graph in a complex manner. In [54], input and communication graph are identical and fixed; in [38], there is also no adaptive communication pattern, as essentially the algorithm may merely decide on how to further separate elements that share the same image under the hash functions applied to them so far.

Various other lower bound techniques exist [34, 75], however, they are not related to the bound presented in Chapter 7. If graph-based, the arguments are often purely information theoretic, in the sense that some information must be exchanged over some bottleneck link or node in a carefully constructed network with diameter larger than two [72, 92]. In our setting, such information theoretic lower bounds will not work: Any two balls may exchange information along $n$ edge-disjoint paths of length two, as the graph describing which edges could *potentially* be used to transmit a message is complete bipartite. In some sense, this is the main contribution of this part of our exposition: We show the existence of a coordination bottleneck in a system without a physical bottleneck.

# Chapter 7

# Lower Bound on Symmetric Balls-into-Bins Algorithms

*"That's probably the most complicated thing I've ever heard." – "Seriously?" – "No, but the way you explain it, it sounds as if it was." – Roger's comment on my first chaotic proof sketch of the balls-into-bins lower bound.*

In this chapter, we derive our lower bound on the parallel complexity of the balls-into-bins problem. We will show that any symmetric balls-into-bins algorithm guaranteeing $\mathcal{O}(n)$ total messages w.h.p. requires at least $(1-o(1))\log^* n$ rounds w.h.p. to achieve a maximal bin load of ${}^{o(\log^* n)}2$. This chapter is based on [69] and the accompanying technical report [68].

## 7.1 Definitions

In fact, our lower bound for the symmetric problem holds for a slightly stronger communication model.

**Problem 7.1** (Acquaintance Balls-into-Bins). *We call an instance of Problem 6.1* acquaintance balls-into-bins problem*, if the following holds. Initially, bins are anonymous, i.e., balls identify bins by u.i.r. port numberings. However, once a ball contacts a bin, it learns its globally unique address, by which it can be contacted reliably. Thus, by means of forwarding addresses, balls can learn to contact specific bins directly. The addresses are abstract in the* *sense that they can be used for this purpose only.*[1] *We call an algorithm solving this problem* acquaintance algorithm.

For this problem, we will show the following result.

**Theorem 7.2.** *Any acquaintance algorithm sending in total $\mathcal{O}(n)$ messages w.h.p. and at most* polylog $n$ *messages per node either incurs a maximal bin load of more than $L \in \mathbb{N}$ w.h.p. or runs for $(1-o(1))\log^* n - \log^* L$ rounds, irrespective of the size of messages.*

In order to show this statement, we need to bound the amount of information a ball can collect during the course of the algorithm. As each ball may contact any bins it has heard of, this information is a subset of an (appropriately labeled) exponentially growing neighborhood of the ball in the graph where edges are created whenever a ball picks a communication partner at random.

**Definition 7.3** (Balls-into-Bins Graph)**.** *The (bipartite and simple)* balls-into-bins graph $G_{\mathcal{A}}(t)$ *associated with an execution of the acquaintance balls-into-bins algorithm $\mathcal{A}$ running for $t \in \mathbb{N}$ rounds is constructed as follows. The node set $V := V_{\sqcup} \cup V_{\circ}$ consists of $|V_{\sqcup}| = |V_{\circ}| = n$ bins and balls. In each round $i \in \{1,\ldots,t\}$, each ball $b \in V_{\circ}$ adds an edge connecting itself to bin $v \in V_{\sqcup}$ if $b$ contacts $v$ by a random choice in that round. By $E_{\mathcal{A}}(i)$ we denote the edges added in round $i$ and $G_{\mathcal{A}}(t) = (V, \cup_{i=1}^{t} E_{\mathcal{A}}(i))$ is the graph containing all edges added until and including round $t$.*

In the remainder of this chapter, we will consider such graphs only.

The proof will argue about certain symmetric subgraphs in which not all balls can decide on bins concurrently without incurring large bin loads. As can be seen by a quick calculation, any connected subgraph containing a cycle is unlikely to occur frequently. For an adaptive algorithm, it is possible that balls make a larger effort in terms of sent messages to break symmetry once they observe a "rare" neighborhood. Therefore, it is mandatory to reason about subgraphs that are trees.

We would like to argue that any algorithm suffers from generating a large number of trees of uniform ball and bin degrees. If we root such a tree at an arbitrary bin, balls cannot distinguish between their parents and children according to this orientation. Thus, they will decide on a bin that is closer to the root with probability inverse proportional to their degree. If bin degrees are by factor $f(n)$ larger than ball degrees, this will result in an expected load of the root of $f(n)$. However, this line of reasoning is too simple. As

[1]This requirement is introduced to prohibit the use of these addresses for symmetry breaking, as is possible for asymmetric algorithms. One may think of the addresses e.g. as being random from a large universe, or the address space might be entirely unknown to the balls.

edges are added to $G$ in different rounds, these edges can be distinguished by the balls. Moreover, even if several balls observe the same local topology in a given round, they may randomize the number of bins they contact during that round, destroying the uniformity of degrees. For these reasons, we (*i*) rely on a more complicated tree in which the degrees are a function of the round number and (*ii*) show that for every acquaintance algorithm a stronger algorithm exists that indeed generates many such trees w.h.p.

In summary, the proof will consist of three steps. Firstly, for any acquaintance algorithm obeying the above bounds on running time and message complexity, an at least equally powerful algorithm from a certain subclass of algorithms exists. Secondly, algorithms from this subclass generate for $(1-o(1))\log^* n$ rounds large numbers of the aforementioned highly symmetric trees in $G_{\mathcal{A}}(t)$ w.h.p. Thirdly, enforcing a decision from all balls in such structures leads to a maximal bin load of $\omega(1)$ w.h.p.

The following definition clarifies what we understand by "equally powerful" in this context.

**Definition 7.4** (W.h.p. Equivalent Algorithms)**.** *We call two Algorithms $\mathcal{A}$ and $\mathcal{A}'$ for Problem 6.1* w.h.p. equivalent *if their output distributions agree on all but a fraction of the events occurring with total probability at most $1/n^c$. That is, if $\Gamma$ denotes the set of possible distributions of balls into bins, we have that*

$$\sum_{\gamma\in\Gamma} |P_{\mathcal{A}}[\gamma] - P_{\mathcal{A}'}[\gamma]| \leq \frac{1}{n^c}.$$

The subclass of algorithms we are interested in is partially characterized by its behaviour on the mentioned subgraphs, hence we need to define the latter first. These subgraphs are special trees, in which all involved balls up to a certain distance from the root see exactly the same topology. This means that (*i*) in each round, all involved balls created exactly the same number of edges by contacting bins randomly, (*ii*) each bin has a degree that depends on the round when it was contacted first only, (*iii*) all edges of such bin are formed in exactly this round, and (*iv*) this scheme repeats itself up to a distance that is sufficiently large for the balls not to see any irregularities that might help in breaking symmetry. These properties are satisfied by the following tree structure.

**Definition 7.5** (Layered $(\Delta^{\sqcup}, \Delta^{\circ}, \mathcal{D})$-Trees)**.** *A* layered $(\Delta^{\sqcup}, \Delta^{\circ}, \mathcal{D})$-tree *of* $\ell \in \mathbb{N}_0$ levels *rooted at bin $R$ is defined as follows, where $\Delta^{\sqcup} = (\Delta_1^{\sqcup}, \ldots, \Delta_\ell^{\sqcup})$ and $\Delta^{\circ} = (\Delta_1^{\circ}, \ldots, \Delta_\ell^{\circ})$ are the vectors of bins' and balls' degrees on different levels, respectively, and $\mathcal{D} \in \mathbb{N}$.*

*If $\ell = 0$, the "tree" is simply a single bin. If $\ell > 0$, the subgraph of $G_{\mathcal{A}}(\ell)$ induced by $\mathcal{N}_R^{(2\mathcal{D})}$ is a tree, where ball degrees are uniformly $\sum_{i=1}^{\ell} \Delta_i^{\circ}$.*

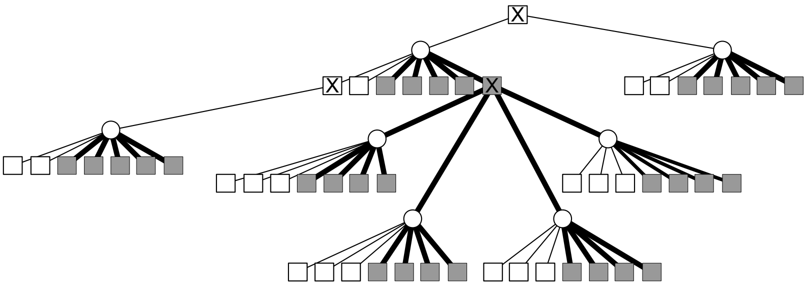

Figure 7.1: Part of a $((2,5),(3,5),\mathcal{D})$-tree rooted at the topmost bin. Bins are squares and balls are circles; neighborhoods of all balls and the bins marked by an "X" are depicted completely, the remainder of the tree is left out. Thin edges and white bins were added to the structure in the first round, thick edges and grey bins in the second. Up to distance $2\mathcal{D}$ from the root, the pattern repeats itself, i.e., the $(2\mathcal{D}-d)$-neighborhoods of all balls up to depth $d$ appear identical.

*Except for leaves, a bin that is added to the structure in round $i \in \{1,\ldots,\ell\}$ has degree $\Delta_i^{\sqcup}$ with all its edges in $E_{\mathcal{A}}(i)$. See Figure 7.1 for an illustration.*

Intuitively, layered trees are crafted to present symmetric neighborhoods to nodes which are not aware of leaves. Hence, if bins' degrees are large compared to balls' degrees, not all balls can decide simultaneously without risking to overload bins. This statement will be made mathematically precise later.

We are now in the position to define the subclass of algorithms we will analyze. The main reason to resort to this subclass is that acquaintance algorithms may enforce seemingly asymmetric structures, which complicates proving a lower bound. In order to avoid this, we grant the algorithms additional random choices, restoring symmetry. The new algorithms must be even stronger, since they have more information available, yet they will generate many layered trees. Since we consider such algorithms specifically for this purpose, this is hard-wired in the definition.

**Definition 7.6** (Oblivious-Choice Algorithms)**.** *Assume that an acquaintance Algorithm $\mathcal{A}$ running for at most $t$ rounds, $\Delta^{\sqcup} = (\Delta_1^{\sqcup},\ldots,\Delta_t^{\sqcup})$, and $\Delta^{\circ} = (\Delta_1^{\circ},\ldots,\Delta_t^{\circ})$ are given. Suppose $T = (T_0,\ldots,T_t) \in (\mathbb{R}^+)^t$ is a sequence such that $\Delta_i^{\circ} \in \mathcal{O}(n/T_{i-1})$ for all $i \in \{1,\ldots,t\}$ and for all $i \in \{0,\ldots,t\}$ the number of disjoint layered $((\Delta_1^{\sqcup},\ldots,\Delta_i^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_i^{\circ}),2^t)$-trees in $G_{\mathcal{A}}(i)$ is at least $T_i$ w.h.p.*

*We call $\mathcal{A}$ a* $(\Delta^{\sqcup}, \Delta^{\circ}, T)$-oblivious-choice algorithm, *if the following requirements are met:*

*(i) The algorithm terminates at the end of round $t$, when all balls simultaneously decide into which bin they are placed. A ball's decision is based on its $2^t$-neighborhood in $G_{\mathcal{A}}(t)$, including the random bits of any node within that distance, and all bins within this distance are feasible choices.*[2]

*(ii) In round $i \in \{1, \dots, t\}$, each ball $b$ decides on a number of bins to contact and chooses that many bins u.i.r., forming the respective edges in $G_{\mathcal{A}}(i)$ if not yet present. This decision may resort to the topology of the $2^t$-hop neighborhood of a ball in $G_{\mathcal{A}}(i-1)$ (where $G_{\mathcal{A}}(0)$ is the graph containing no edges).*

*(iii) In round $i \in \{1, \dots, t\}$, it holds w.h.p. that all balls in depth $d \leq 2^t$ of $\Omega(T_{i-1})$ layered $((\Delta_1^{\sqcup}, \dots, \Delta_{i-1}^{\sqcup}), (\Delta_1^{\circ}, \dots, \Delta_{i-1}^{\circ}), 2^t)$-trees in $G_{\mathcal{A}}(i)$ each choose $\Delta_i^{\circ}$ bins to contact.*

The larger $t$ can be, the longer it will take until eventually no more layered trees occur and all balls may decide safely.

## 7.2 Proof of the Lower Bound

We need to show that for appropriate choices of parameters and non-trivial values of $t$, indeed oblivious-choice algorithms exist. Essentially, this is a consequence of the fact that we construct trees: When growing a tree, each added edge connects to a node outside the tree, therefore leaving a large number of possible endpoints for the edge; in contrast, closing a circle in a small subgraph is unlikely.

**Lemma 7.7.** *Let $\Delta_1^{\circ} \in \mathbb{N}$ and $C > 0$ be constants, $L, t \in \mathbb{N}$ arbitrary, $T_0 := n/(100(\Delta_1^{\circ})^2(2C+1)L) \in \Theta(n/L)$, and $\Delta_1^{\sqcup} := 2L\Delta_1^{\circ}$. Define for $i \in \{2, \dots, t\}$ that*

$$\begin{aligned}\Delta_i^{\circ} &:= \left\lceil \frac{\Delta_1^{\circ} n}{T_{i-1}} \right\rceil, \\ \Delta_i^{\sqcup} &:= 2L\Delta_i^{\circ},\end{aligned}$$

*and for $i \in \{1, \dots, t\}$ that*

$$T_i := 2^{-(n/T_{i-1})^{4 \cdot 2^t}} n.$$

[2] This is a superset of the information a ball can get when executing an acquaintance algorithm, since by address forwarding it might learn of and contact bins up to that distance. Note that randomly deciding on an unknown bin here counts as contacting it, as a single round makes no difference with respect to the stated lower bound.

*If $T_t \in \omega(\sqrt{n}\log n)$ and $n$ is sufficiently large, then any algorithm fulfilling the prerequisites (i), (ii), and (iii) from Definition 7.6 with regard to these parameters that sends at most $Cn^2/T_{i-1}$ messages in round $i \in \{1,\ldots,t\}$ w.h.p. is a $(\Delta^{\sqcup},\Delta^{\circ},T)$-oblivious-choice algorithm.*

*Proof.* Since by definition we have $\Delta_i^{\circ} \in \mathcal{O}(n/T_{i-1})$ for all $i \in \{1,\ldots,t\}$, in order to prove the claim we need to show that at least $T_i$ disjoint layered $((\Delta_1^{\sqcup},\ldots,\Delta_i^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_i^{\circ}),2^t)$-trees occur in $G_{\mathcal{A}}(i)$ w.h.p. We prove this statement by induction. Since $T_0 \leq n$ and every bin is a $((),(),2^t)$-tree, we need to perform the induction step only.

Hence, assume that for $i-1 \in \{0,\ldots,t-1\}$, $T_{i-1}$ lower bounds the number of disjoint layered $((\Delta_1^{\sqcup},\ldots,\Delta_{i-1}^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_{i-1}^{\circ}),2^t)$-trees in $G_{\mathcal{A}}(i-1)$ w.h.p. In other words, the event $\mathcal{E}_1$ that we have at least $T_{i-1}$ such trees occurs w.h.p.

We want to lower bound the probability $p$ that a so far isolated bin $R$ becomes the root of a $((\Delta_1^{\sqcup},\ldots,\Delta_i^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_i^{\circ}),2^t)$-tree in $G_{\mathcal{A}}(i)$. Starting from $R$, we construct the $2\mathcal{D}$-neighborhood of $R$. All involved balls take part in disjoint $((\Delta_1^{\sqcup},\ldots,\Delta_{i-1}^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_{i-1}^{\circ}),2^t)$-trees, all bins incorporated in these trees are not adjacent to edges in $E_{\mathcal{A}}(i)$, and all bins with edges on level $i$ have been isolated until and including round $i-1$.

As the algorithm sends at most $\sum_{j=1}^{i-1} Cn^2/T_{j-1}$ messages until the end of round $i-1$ w.h.p., the expected number of isolated bins after round $i-1$ is at least

$$\begin{aligned}\left(1-\frac{1}{n^c}\right)n\left(1-\frac{1}{n}\right)^{Cn\sum_{j=1}^{i-1} n/T_{j-1}} &\in ne^{-(1+o(1))Cn/T_{i-1}}\\ &\subset ne^{-\mathcal{O}(n/T_{t-1})}\\ &\subset \omega(\log n).\end{aligned}$$

Thus Lemma 2.15 and Corollary 2.13 imply that the event $\mathcal{E}_2$ that at least $ne^{-(1+o(1))Cn/T_{i-1}}$ such bins are available occurs w.h.p.

Denote by $N$ the total number of nodes in the layered tree. Adding balls one by one, in each step we choose a ball out of w.h.p. at least $T_{i-1}-N+1$ remaining balls in disjoint $((\Delta_1^{\sqcup},\ldots,\Delta_{i-1}^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_{i-1}^{\circ}),2^t)$-trees, connect it to a bin already in the tree, and connect it to $\Delta_i^{\circ}-1$ of the w.h.p. at least $ne^{-(1+o(1))Cn/T_{i-1}}-N+1$ remaining bins that have degree zero in $G_{\mathcal{A}}(i-1)$. Denote by $\mathcal{E}_3$ the event that the tree is constructed successfully and let us bound its probability.

Observe that because for all $i \in \{1,\ldots,t\}$ we have that $\Delta_i^{\sqcup} > 2\Delta_{i-1}^{\sqcup}$ and $\Delta_i^{\circ} > 2\Delta_{i-1}^{\circ}$, it holds that

$$N < \sum_{d=0}^{2^t}\left(\Delta_i^{\sqcup}\left(\sum_{j=1}^{i}\Delta_j^{\circ}\right)\right)^d < \sum_{d=0}^{2^t}\left(2\Delta_i^{\sqcup}\Delta_i^{\circ}\right)^d < 2\left(2\Delta_i^{\sqcup}\Delta_i^{\circ}\right)^{2^t}. \tag{7.1}$$

Furthermore, the inductive definitions of $\Delta_i^{\sqcup}$, $\Delta_i^{\circ}$, and $T_i$, the prerequisite that $T_t \in \omega(\sqrt{n}\log n)$, and basic calculations reveal that for all $i \in \{1,\dots,t\}$, we have the simpler bound of

$$N < 2\left(2\Delta_i^{\sqcup}\Delta_i^{\circ}\right)^{2^t} < 2(4L+1)^{2t}\left(\frac{\Delta_1^{\sqcup}n}{T_{i-1}}\right)^{4t} \in ne^{-\omega(n/T_{i-1})} \cap o(T_{i-1}) \quad (7.2)$$

on $N$.

Recall that $\mathcal{A}$ is an oblivious-choice algorithm, i.e., the bins that are contacted in a given round are chosen independently. Thus, provided that $\mathcal{E}_1$ occurs, the (conditional) probability that a bin that has already been attached to its parent in the tree is contacted by the first random choice of exactly $\Delta_i^{\sqcup}-1$ balls that are sufficiently close to the roots of disjoint $((\Delta_1^{\sqcup},\dots,\Delta_{i-1}^{\sqcup}),(\Delta_1^{\circ},\dots,\Delta_{i-1}^{\circ}),2^t)$-trees is lower bounded by

$$\begin{aligned}
&\binom{T_{i-1}-N+(\Delta_i^{\sqcup}-1)}{\Delta_i^{\sqcup}-1}\left(\frac{1}{n}\right)^{\Delta_i^{\sqcup}-1}\left(1-\frac{1}{n}\right)^{(\Delta_i^{\sqcup}-1)(\Delta_i^{\circ}-1)}\\
\overset{(7.2)}{\in}\ &\left(\frac{T_{i-1}}{n\Delta_i^{\sqcup}}\right)^{(1+o(1))(\Delta_i^{\sqcup}-1)}.
\end{aligned}$$

Because $\Delta_i^{\sqcup} \in \mathcal{O}(n/T_{i-1})$, it holds that $\ln(n\Delta_i^{\sqcup}/T_{i-1}) \in o(n/T_{i-1})$. Thus, going over all bins (including the root, where the factor in the exponent is $\Delta_i^{\sqcup}$ instead of $\Delta_i^{\sqcup}-1$), we can lower bound the probability that all bins are contacted by the right number of balls by

$$\left(\frac{T_{i-1}}{n\Delta_i^{\sqcup}}\right)^{(1+o(1))N} \in e^{-(1+o(1))Nn/T_{i-1}},$$

as less than $N$ balls need to be added to the tree in total. Note that we have not made sure yet that the bins are not contacted by other balls; $\mathcal{E}_3$ is concerned with constructing the tree as a subgraph of $G_{\mathcal{A}}(t)$ only.

For $\mathcal{E}_3$ to happen, we also need that all balls that are added to the tree contact previously isolated bins. Hence, in total fewer than $N$ u.i.r. choices need to hit different bins from a subset of size $ne^{-(1+o(1))Cn/T_{i-1}}$. This probability can be bounded by

$$\left(\frac{ne^{-(1+o(1))Cn/T_{i-1}}-N}{n}\right)^{N} \overset{(7.2)}{\in} e^{-(1+o(1))CNn/T_{i-1}}.$$

Now, after constructing the tree, we need to make sure that it is indeed the induced subgraph of $\mathcal{N}_R^{(2\mathcal{D})}$ in $G_{\mathcal{A}}(i)$, i.e., no further edges connect to any nodes in the tree. Denote this event by $\mathcal{E}_4$. As we already "used" all edges of balls inside the tree and there are no more than $Cn^2/T_{i-1}$ edges created by balls outside the tree, $\mathcal{E}_4$ happens with probability at least

$$\left(1-\frac{N}{n}\right)^{Cn^2/T_{i-1}} \in e^{-(1+o(1))CNn/T_{i-1}}.$$

Combining all factors, we obtain that

$$\begin{aligned}
p &\geq P[\mathcal{E}_1]\cdot P[\mathcal{E}_2\,|\,\mathcal{E}_1]\cdot P[\mathcal{E}_3\,|\,\mathcal{E}_1\wedge\mathcal{E}_2]\cdot P[\mathcal{E}_4\,|\,\mathcal{E}_1\wedge\mathcal{E}_2\wedge\mathcal{E}_3]\\
&\in \left(1-\frac{1}{n^c}\right)^2 e^{-(1+o(1))(C+1)Nn/T_{i-1}}e^{-(1+o(1))CNn/T_{i-1}}\\
&= e^{-(1+o(1))(2C+1)Nn/T_{i-1}}\\
&\overset{(7.1)}{\subseteq} 2Ne^{-(1+o(1))(2C+1)(2\Delta_i^{\sqcup}\Delta_i^{\circ})^{2^t}n/T_{i-1}}e^{(1+o(1))Cn/T_{i-1}}\\
&\subseteq 2Ne^{-(1+o(1))(2C+1)\left(4L(2\Delta_1^{\circ}n/T_{i-1})^2\right)^{2^t}n/T_{i-1}}e^{(1+o(1))Cn/T_{i-1}}\\
&\subseteq 2N2^{-\left(n/T_0(n/T_{i-1})^3\right)^{2^t}}e^{(1+o(1))Cn/T_{i-1}}\\
&\subseteq \frac{2NT_i}{n}e^{(1+o(1))Cn/T_{i-1}}.
\end{aligned}$$

We conclude that the expected value of the random variable $X$ counting the number of disjoint $((\Delta_1^{\sqcup},\ldots,\Delta_i^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_i^{\sqcup}),2^t)$-trees is lower bounded by $\mathbb{E}[X] > 2T_i$, as at least $e^{-(1+o(1))Cn/T_{i-1}}n$ isolated bins are left that may serve as root of (not necessarily disjoint) trees and each tree contains less than $N$ bins.

Finally, having fixed $G_{\mathcal{A}}(i-1)$, $X$ becomes a function of w.h.p. at most $\mathcal{O}(n^2/T_{i-1}) \subseteq \mathcal{O}(n^2/T_{t-1}) \subseteq \mathcal{O}(n\log(n/T_t)) \subseteq \mathcal{O}(n\log n)$ u.i.r. chosen bins contacted by the balls in round $i$. Each of the corresponding random variables may change the value of $X$ by at most three: An edge insertion may add one tree or remove two, while deleting an edge removes at most one tree and creates at most two. Due to the prerequisite that $T_i \geq T_t \in \omega(\sqrt{n}\log n)$, we have $\mathbb{E}[X] \in \omega(\sqrt{n}\log n)$. Hence we can apply Theorem 2.17 in order to obtain

$$P\left[X < \frac{\mathbb{E}[X]}{2}\right] \in e^{-\Omega\left(\mathbb{E}[X]^2/(n\log n)\right)} \subseteq n^{-\omega(1)},$$

proving the statement of the lemma. □

We see that the probability that layered trees occur falls at most exponentially in their size to the power of $4\cdot 2^t$. Since $t$ is very small, i.e., smaller than $\log^* n$, this rate of growth is comparable to exponentiation by a polynomial in the size of the tree. Therefore, one may expect that the requirement of $T_t \in \omega(\sqrt{n}/\log n)$ can be maintained for values of $t$ in $\Omega(\log^* n)$. Calculations reveal that even $t \in (1-o(1))\log^* n$ is feasible.

**Lemma 7.8.** *Using the notation of Lemma 7.7, it holds for*

$$t \leq t_0(n, L) \in (1 - o(1)) \log^* n - \log^* L$$

*that* $T_t \in \omega(\sqrt{n}/\log n)$.

*Proof.* By basic calculus. We refer to [68]. □

In light of the upper bounds we will show in the next chapter, this interplay between $L$ and $t$ is by no means arbitrary. We will see that if for any $r \in \mathbb{N}$ one accepts a maximal bin load of $\log^{(r)} n/\log^{(r+1)} n + r + \mathcal{O}(1)$, Problem 6.1 can be solved in $r + \mathcal{O}(1)$ rounds.

Since now we know that critical subgraphs occur frequently for specific algorithms, next we prove that this subclass of algorithms is as powerful as acquaintance algorithms of certain bounds on time and message complexity.

**Lemma 7.9.** *Suppose the acquaintance Algorithm $\mathcal{A}$ solves Problem 6.1 within $t \leq t_0(n, L)$, $L \in \mathbb{N}$, rounds w.h.p. ($t_0$ as in Lemma 7.8), sending w.h.p. at most $\mathcal{O}(n)$ messages in total and* polylog $n$ *messages per node. Then, for sufficiently large $n$, a constant $\Delta_1^\circ$ and an oblivious-choice algorithm $\mathcal{A}'$ with regard to the set of parameters specified in Lemma 7.7 exists that sends at most $\mathcal{O}(n^2/T_{i-1})$ messages in round $i \in \{1, \ldots, t\}$ w.h.p., terminates at the end of round $t$, and is w.h.p. equivalent to $\mathcal{A}$.*

*Proof.* Observe that $\mathcal{A}$ has only two means of disseminating information: Either, balls can randomly connect to unknown bins, or they can send information to bins known from previous messages. Thus, any two nodes at distance larger than $2^t$ from each other in $G_{\mathcal{A}}(i-1)$ must act independently in round $i$. Since degrees are at most polylog $n$ w.h.p., w.h.p. no ball knows more than $(\text{polylog}\, n)^{2^t} \subseteq n^{o(1)}$ bins (w.l.o.g. $t_0(n, l) \leq \log^* n - 2$). Assume that in a given round a ball chooses $k$ bins, excluding the ones of which he already obtained the global address. If it contacted $k' := \lceil 3ck \rceil$ bins u.i.r. and dropped any drawn bin that it already knows (including repetitions in the current round), it would make with probability at most

$$\sum_{j=k'-k+1}^{k'} \binom{k'}{j} \left(1 - \frac{1}{n^{1-o(1)}}\right)^{k'-j} \frac{1}{n^{(1-o(1))j}} \subset n^{-2(1-o(1))c} \text{polylog}\, n \subset n^{-c}$$

less than $k$ new contacts. Thus, we may modify $\mathcal{A}$ such that it chooses $\mathcal{O}(k)$ bins u.i.r. whenever it would contact $k$ bins randomly. This can be seen as augmenting $G_{\mathcal{A}}(i)$ by additional edges. By ignoring these edges, the resulting algorithm $\mathcal{A}'$ is capable of (locally) basing its decisions on the probability distribution of graphs $G_{\mathcal{A}}(i)$. Hence, Condition (*ii*) from Definition 7.6 is met by the modified algorithm.

Condition $(i)$ forces $\mathcal{A}'$ to terminate in round $t$ even if $\mathcal{A}$ does not. However, since $\mathcal{A}$ must terminate in round $t$ w.h.p., balls may choose arbitrarily in this case, w.h.p. not changing the output compared to $\mathcal{A}$. On the other hand, we can certainly delay the termination of $\mathcal{A}$ until round $t$ if $\mathcal{A}$ would terminate earlier, without changing the results. Thus, it remains to show that we can further change the execution of $\mathcal{A}$ during the first $t$ rounds in a way ensuring Condition $(iii)$ of the definition, while maintaining the required bound on the number of messages.

To this end, we modify $\mathcal{A}$ inductively, where again in each round for some balls we increase the number of randomly contacted bins compared to an execution of $\mathcal{A}$; certainly this will not affect Conditions $(i)$ and $(ii)$ from Definition 7.6, and $\mathcal{A}'$ will exhibit the same output distribution as $\mathcal{A}$ (up to a fraction of $1/n^c$ of the executions) if $\mathcal{A}'$ ignores any of the additional edges when placing the balls at the end of round $t$.

Now, assume that the claim holds until round $i-1 \in \{0,\ldots,t-1\}$. In round $i \in \{1,\ldots,t\}$, any pair of balls in depth at most $2^t$ of disjoint $((\Delta_1^{\sqcup},\ldots,\Delta_{i-1}^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_{i-1}^{\circ}),2^t)$-trees in $G_{\mathcal{A}'}(i)$ are in distance at least $2^{t+1}$ from each other, i.e., they must decide mutually independently on the number of bins to contact. Consequently, since $\mathcal{A}$ has less information at hand than $\mathcal{A}'$, these balls would also decide independently in the corresponding execution of $\mathcal{A}$. For sufficiently large $n$, (the already modified variant of) $\mathcal{A}$ will send at most $Cn$ messages w.h.p. (for some constant $C \in \mathbb{R}^+$). Hence, the balls that are up to depth $2^t$ of such a tree send together in expectation less than $2nC/T_{i-1}$ messages, since otherwise Corollary 2.13 and the fact that $T_{i-1} \in \omega(\log n)$ would imply that at least $(2-o(1))Cn$ messages would be sent by $\mathcal{A}$ in total w.h.p. Consequently, by Markov's Bound (Theorem 2.6), with independent probability at least $1/2$, all balls in depth $2^t$ or less of a layered $((\Delta_1^{\sqcup},\ldots,\Delta_{i-1}^{\sqcup}),(\Delta_1^{\circ},\ldots,\Delta_{i-1}^{\circ}),2^t)$-tree in union send no more than $4Cn/T_{i-1}$ messages in round $i$. Using Corollary 2.13 again, we conclude that for $\Omega(T_{i-1})$ many trees it holds that none of the balls in depth at most $2^t$ will send more than $4Cn/T_{i-1}$ messages to randomly chosen bins w.h.p.

When executing $\mathcal{A}'$, we demand that each such ball randomly contacts *exactly* that many bins, i.e., with $\Delta_1^{\circ} := 4C$ Condition $(iii)$ of Definition 7.6 is met. By Corollary 2.13, this way at most $\mathcal{O}(n + n^2/T_{i-1}) = \mathcal{O}(n^2/T_{i-1})$ messages are sent in round $i$ w.h.p., as claimed. Moreover, the new algorithm can ensure to follow the same probability distribution of bin contacts as $\mathcal{A}$, simply by ignoring the additional random choices made. This completes the induction step and thus the proof. $\square$

The final ingredient is to show that randomization is insufficient to deal with the highly symmetric topologies of layered trees. In particular, balls that decide on bins despite not being aware of leaves cannot avoid risking to choose the root bin of the tree. If all balls in a tree where bin degrees are large compared to ball degrees decide, this results in a large load of the root bin.

**Lemma 7.10.** *Suppose after $t$ rounds of some Algorithm $\mathcal{A}$ ball $b$ is in depth at most $2^t$ of a layered $(\Delta^{\sqcup}, \Delta^{\circ}, 2^t)$-tree of $t$ levels in $G_{\mathcal{A}}(t)$. We fix the topology of the layered tree. Let $v$ be a bin in distance $d \leq 2^t$ from $b$ in $G_{\mathcal{A}}(t)$ and assume that the edge sequence of the (unique) shortest path from $b$ to $v$ is $e_1, \ldots, e_d$. If $b$ decides on a bin in round $t$, the probability that $b$ places itself in $v$ depends on the sequence of rounds $\ell_1, \ldots, \ell_d \in \{1, \ldots, t\}^d$ in which the edges $e_1, \ldots, e_d$ have been created only.*

*Proof.* Observe that since $b$ is a ball, it must have an odd distance from the root of the layered $(\Delta^{\sqcup}, \Delta^{\circ}, 2^t)$-tree it participates in. Thus, the $2^t$-neighborhood of $b$ is a subset of the $(2^{t+1} - 1)$-neighborhood of the root of the layered tree. Therefore, this neighborhood is a balanced tree of uniform ball degrees. Moreover, for all $i \in \{1, \ldots, t\}$, the number of edges from $E_{\mathcal{A}}(i)$ balls up to distance $2^t$ from $b$ are adjacent to is the same. Bin degrees depend on the round $i$ in which they have been contacted first only, and all edges of a bin in the tree were created in the respective round (cf. Figure 7.1).

Let $b_1, \ldots, b_n$ and $v_1, \ldots, v_n$ be global, fixed enumerations of the balls and bins, respectively. Fix a topology $T$ of the $2^t$-neighborhood of $b$ with regard to these enumerations. Assume that $v$ and $w$ are two bins in $T$ for which the edges on the shortest paths from $b$ to $v$ resp. $w$ were added in the rounds $\ell_1, \ldots, \ell_d$, $d \in \{1, \ldots, 2^t\}$. Assume that $x$ and $y$ are the first distinct nodes on the shortest paths from $b$ to $v$ and $w$, respectively. The above observations show that the subtrees of $T$ rooted at $x$ and $y$ are isomorphic (cf. Figure 7.2). Thus, a graph isomorphism $f$ exists that "exchanges" the two subtrees (preserving their orientation), fixes all other nodes, and fulfills that $f(v) = w$ and $f \circ f$ is the identity. We choose such an $f$ and fix it. Denote by $p(b_i, v_j) \in \{1, \ldots, n\}$, $i, j \in \{1, \ldots, n\}$, the port number $v_j$ has in the port numbering of $b_i$ and by $p(v_i, b_j)$ the number $b_j$ has in the port numbering of bin $v_i$. Similarly, $r(b_i)$ and $r(v_i)$, $i \in \{1, \ldots, n\}$, denote the string of random bits $b_i$ and $v_i$ use for randomization, respectively. Using $f$, we define the automorphism $h : S \to S$ on the set of tuples of possible port numberings and random strings $(p(\cdot, \cdot), r(\cdot))$ by

$$h((p(\cdot, \cdot), r(\cdot))) := (p(f(\cdot), f(\cdot)), r(f(\cdot))).$$

Set

$$S_v := \{(p(\cdot, \cdot), r(\cdot)) \,|\, T \text{ occurs} \wedge b \text{ chooses } v\} \subset S$$

and $S_w$ analogously. We claim that $h(S_v) = S_w$ (and therefore also $h(S_w) = h^2(S_v) = S_v$). This means when applying $h$ to an element of $S_v$, the topology

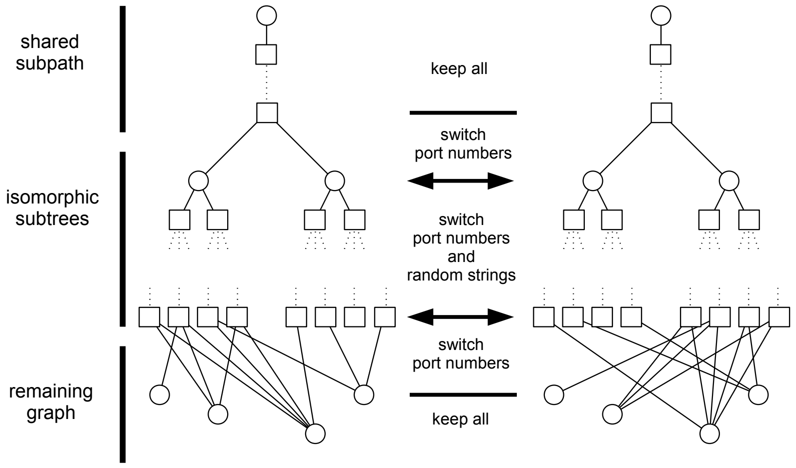

Figure 7.2: Example for the effect of $h$ on the topology of $G_{\mathcal{A}}(t)$. Switching port numberings and random labels of isomorphic subtrees and the port numberings of their neighborhood, nodes in these subtrees essentially "switch their identity" with their counterparts. Since the local view of the topmost ball is completely contained within the tree, it cannot distinguish between the two configurations.

$T$ is preserved and $b$ chooses $w$ instead of $v$. To see this, observe that $\mathcal{A}$ can be interpreted as deterministic algorithm on the (randomly) labeled graph where nodes $u$ are labeled $r(u)$ and edges $(u, u')$ are labeled $(p(u, u'), p(u', u))$. Hence, $h$ simply switches the local views of $u$ and $f(u)$ in that graph and a node $u$ takes the role of $f(u)$ and vice versa (cf. Figure 7.2). Thus, $b$ will choose $f(v) = w$ in the execution with the new labeling. On the other hand, $T$ is preserved because we chose the function $f$ in a way ensuring that we mapped the two subtrees in $T$ that are rooted at $x$ and $y$ to each other by a graph isomorphism, i.e., the topology with regard to the fixed enumerations $b_1, \ldots, b_n$ and $v_1, \ldots, v_n$ did not change.

In summary, for any topology $T$ of the $2^t$-neighborhood of $b$ in a layered $(\Delta^{\sqcup}, \Delta^{\circ}, 2^t)$-tree such that $b$ is connected to $v$ and $w$ by shortest paths for which the sequences of round numbers when the edges were created are the same, we have that $S_v = h(S_w)$. Since both port numberings and random inputs are chosen independently, we conclude that $P[(p(\cdot, \cdot), r(\cdot)) \in S_v] =$

$P[(p(\cdot,\cdot),r(\cdot)) \in S_w]$, i.e., $b$ must choose $v$ and $w$ with equal probability as claimed. □

We are now in the position to prove our lower bound on the trade-off between maximal bin load and running time of acquaintance algorithms.

*Proof of Theorem 7.2.* Assume that Algorithm $\mathcal{A}$ solves Problem 7.1 within at most $t \leq t_0 \in (1-o(1))\log^* n - \log^* L$ rounds w.h.p. ($t_0$ as in Lemma 7.8). Thus, due to Lemma 7.9 a constant $\Delta_1^\circ$ and an oblivious-choice Algorithm $\mathcal{A}'$ with regard to the parameters from Lemma 7.7 whose maximal bin load is w.h.p. the same as the one of $\mathcal{A}$ exist. In the following, we use the notation from Lemma 7.7.

Suppose that $R$ is the root bin of a layered $(\Delta^\sqcup, \Delta^\circ, 2^t)$-tree in $G_{\mathcal{A}'}(t)$. According to Lemma 7.10, for all balls $b$ in distance up to $2^t$ from $R$, the probability $p$ to choose a bin $v$ solely depends on the sequence $s(b,v) = (s_1,\ldots,s_d)$ of round numbers when the edges on the shortest path from $R$ to $b$ were created. Set $S := \bigcup_{i=1}^{2^{t-1}} S_{2i-1}$, where $S_d$ denotes the set of round sequences $s = (s_1,\ldots,s_d)$ of (odd) length $d(s) := d$ from balls to bins (inside the tree). Denote for $s \in S$ by $p(s)$ the probability that a ball $b$ within distance $2^t$ from $R$ decides on (any) bin $v$ with $s(b,v) = s$ and by $X$ the random variable counting the number of balls deciding on $R$. Recall that for any $i \in \{1,\ldots,t\}$, we have $\Delta_i^\sqcup = 2L\Delta_i^\circ$. We compute

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{s\in S} p(s)\frac{|\{b \in V_\circ \mid s(b,R) = s\}}{|\{v \in V_\sqcup \mid s(b,v) = s\}|} \\
&= \sum_{s\in S} p(s)\frac{\Delta_{s_1}^\sqcup \prod_{i=1}^{\lfloor d(s)/2\rfloor} \Delta_{s_{2i}}^\circ \Delta_{s_{2i+1}}^\sqcup}{\Delta_{s_1}^\circ \prod_{i=1}^{\lfloor d(s)/2\rfloor} \Delta_{s_{2i}}^\sqcup \Delta_{s_{2i+1}}^\circ} \\
&= \sum_{s\in S} p(s)2L \\
&= 2L,
\end{aligned}$$

as each ball must decide with probability 1 on a bin within distance $2^t$ (Condition $(ii)$ of Definition 7.6).

On the other hand, the maximal possible load of $R$ is the number of balls up to depth $2^t$ of the tree, which we observed to be less than $(2\Delta_t^\sqcup\Delta_t^\circ)^{2^t} \in \mathcal{O}((n/T_{t-1})^2) \subset o(\log^2 n)$. We infer that for sufficiently large $n$ we have that $P[X > L] > 1/\log^2 n$, since otherwise

$$2L = \mathbb{E}[X] \leq (1 - P[X > L])L + P[X > L]\log^2 n < 2L.$$

As Lemma 7.8 states that the number of disjoint layered $(\Delta^\sqcup, \Delta^\circ, 2^t)$-trees is at least $T_t \in \omega(\sqrt{n}/\log n)$ w.h.p., we have for the random variable $Y$

counting the number of roots of such trees that get a bin load of more than $L$ that

$$\mathbb{E}(Y) \geq \left(1 - \frac{1}{n^c}\right) P[X > L] T_t \in \omega(\log n).$$

Recall that Lemma 7.10 holds for fixed topologies of the trees, i.e., the estimates for $P[X > L]$ and thus $\mathbb{E}(Y)$ follow after fixing the topology up to distance $2^{t+1}$ from all the roots first. Thus, the bound on the probability that a root bin gets a load of more than $L$ is independent of the other roots' loads, since there is no communication between the involved balls. We conclude that we can apply Corollary 2.13 to $Y$ in order to see that $Y > 0$ w.h.p., i.e., $\mathcal{A}'$ incurs a maximal bin load larger than $L$ w.h.p. Because $\mathcal{A}$ and $\mathcal{A}'$ are w.h.p. equivalent, the same holds true for $\mathcal{A}$, proving the claim. □

### 7.2.1 Generalizations

For ease of presentation, the proof of the lower bound assumed that bins do not contact other bins. This is however not necessary.

**Corollary 7.11.** *Theorem 7.2 holds also if bins may directly exchange messages.*

*Proof Sketch.* The presented technique is sufficient for the more general case, as can be seen by the following reasoning. To adapt the proof, we have to consider trees similar to layered $(\Delta^{\circ}, \Delta^{\sqcup}, 2^t)$-trees, where now also bins form edges. Therefore, also bins may create an in each round exponentially growing number of edges to other bins. However, the probability that a bin is the root of such a tree structure in round $i$ will still be lower bounded by $2^{-(n/T_{i-1})^{f(t)}}$, where $f(t)$ is some function such that $\log^*(f(t)) \in \log^* t + \mathcal{O}(1)$ and $T_{i-1}$ is a lower bound on the number of such roots in round $i-1$. Hence, Lemmas 7.7 and 7.8 can be changed accordingly. From that point on, the remainder of the proof can be carried out analogously. □

It is important to be aware that this holds only as long as bins initially identify each other according to u.i.r. port numberings as well. If bins are aware of a globally consistent labeling of all bins, an asymmetric algorithm can be executed, as bins may support balls in doing asymmetric random choices.

Similarly, the upper bound on the number of messages individual nodes send can be relaxed.

**Corollary 7.12.** *Theorem 7.2 holds also if nodes send at most $\lambda n$ messages in total, where $\lambda \in [0, 1)$ is a constant.*

*Proof Sketch.* The critical point of the argumentation is in the proof of Lemma 7.9, where we replace nodes' new contacts by u.i.r. chosen bins. For large numbers of messages, we can no longer guarantee that increasing the number of balls' random choices by a constant factor can w.h.p. compensate for the fact that balls will always contact different bins with each additional message. Rather, we have to distinguish between nodes sending many messages and ones sending only few (e.g. $\operatorname{polylog} n$). Only to the latter we apply the replacement scheme.

Of course, this introduces new difficulties. For instance, we need to observe that still a constant fraction of the bins remains untouched by balls sending many messages for the proof of Lemma 7.7 to hold. The worst case here would be that $\mathcal{O}(1)$ nodes send $\lambda n$ messages during the course of the algorithm, since the bound of $\mathcal{O}(n)$ total messages w.h.p. must not be violated. Thus, the probability that a bin is not contacted by such a ball is lower bounded by

$$1-(1-\lambda)^{\mathcal{O}(1)} \subseteq \Omega(1).$$

Using standard techniques, this gives that still many bins are never contacted during the course of the algorithm w.h.p. Similarly, we need to be sure that sufficiently many of the already constructed trees are not contacted by "outsiders"; here we get probabilities that fall exponentially in the size of such a tree, which is sufficient for the applied techniques.

Another aspect is that now care has to be taken when applying Theorem 2.17 to finish the proof of Lemma 7.7. The problem here is that the random variable describing the edges formed by a ball of large degree is not the product of independent random variables. On the other hand, treating it as a single variable when applying the theorem, it might affect all of the layered trees, rendering the bound from the theorem useless. Thus, we resort to first observing that not too many nodes will send a lot of messages, then fixing their random choices, subsequently bounding the expected number of layered trees conditional to these choices already being made, and finally applying Theorem 2.17 merely to the respective random variable depending on the edges created u.i.r. by balls with small degrees only. □

Note that if we remove the upper bound on the number of messages a single node might send entirely, there is a trivial solution:

1. With probability, say, $1/\sqrt{n}$, a ball contacts $\sqrt{n}$ bins.

2. These balls perform a leader election on the resulting graph (using random identifiers).

3. Contacting all bins, the leader coordinates a perfect distribution of the balls.

In the first step $\mathcal{O}(n)$ messages are sent w.h.p. Moreover, the subgraph induced by the created edges is connected and has constant diameter w.h.p. Hence step 2, which can be executed with $\mathcal{O}(n)$ messages w.h.p., will result in a single leader within a constant number of rounds, implying that step 3 requires $\mathcal{O}(n)$ messages as well. However, this algorithm introduces a central coordination instance. If this was a feasible solution, there would be no need for a parallel balls-into-bins algorithm in the first place.

In light of these results, our lower bound essentially boils down to the following. Any acquaintance algorithm that guarantees w.h.p. both small bin loads and asymptotically optimal $\mathcal{O}(n)$ messages requires $(1-o(1))\log^* n$ rounds.

# Chapter 8

# Balls-into-Bins Algorithms

*"I'm getting too old for this. I wish one day I'd manage to finish* earlier *than the night of the deadline." – Thomas Locher right before submitting our first joint paper, at about 11 pm on a Friday.*

In this chapter, which is also based on [68, 69], we will match the lower bound from the previous one. More precisely, we show that (*i*) a symmetric algorithm can achieve a constant bin load in $\log^* n + \mathcal{O}(1)$ time with $\mathcal{O}(n)$ messages w.h.p. and (*ii*) if we drop either of the requirements of symmetry, constant maximal bin load, or $\mathcal{O}(n)$ total messages, a constant-time solution exists. Moreover, we briefly present an application of the given techniques to an exemplary routing problem.

## 8.1 Optimal Symmetric Algorithm

Our basic symmetric Algorithm $\mathcal{A}_b$ originates from a very simple idea. Assume that all balls behave identically. Given the constraint that we want to guarantee a message complexity of $\mathcal{O}(n)$, it is infeasible that a ball sends more than constantly many messages in the first round. Hence, let each ball contact one random bin. The resulting distribution is well known; in particular, w.h.p. a fraction of $1 - 1/e \pm o(1)$ of the bins will receive at least one message. Since we strive for small bin loads, suppose each bin chooses one of the balls it contacted to be placed into it.

A non-adaptive algorithm would be forced to place each ball into the single bin it contacted, implying a roughly logarithmic maximal bin load. Being adaptive, we can instead let each ball that could not be successfully placed try again by approaching a different bin. However, we can do better. As we know that no more than $(1 + o(1))n/e$ balls remain w.h.p., each such ball may try out *two* random bins, still guaranteeing that we have less than *$n$ requests* in the second round. If again each bin accepts one request, the probability of failure will be much smaller.

Roughly speaking, we can argue more generally as follows. The probability of a ball not being placed in a round where it sends $k$ requests is $2^{-\Omega(k)}$. Using Chernoff's bound, the number of remaining balls thus drops by (almost) this factor, enabling us to increase the number of requests comparably. Overall, the number of requests in round $i$ grows like ${}^{\Omega(i)}2$ until it becomes e.g. $\log n$. By then, each ball will be placed within a constant number of rounds w.h.p.

Formally, initialized with $k(1) := 1$ and $i := 1$, Algorithm $\mathcal{A}_b$ executes the following loop until termination:

1. Balls contact $\lfloor k(i) \rfloor$ u.i.r. bins, requesting permission to be placed into them.

2. Each bin accepts one of the requests (if it received any) and notifies the respective ball.

3. Any ball receiving at least one acceptance notification chooses an arbitrary of the respective bins to be placed into it and terminates.

4. Set $k(i+1) := \min\{k(i)e^{\lfloor k(i) \rfloor/5}, \sqrt{\log n}\}$ and $i := i + 1$.

We will refer to a single execution of the loop of $\mathcal{A}_b$ (or our later algorithms) as a *phase*.

Since in each phase of $\mathcal{A}_b$ some messages will reach their destination, the algorithm will eventually terminate. To give strong bounds on its running time, however, we need some helper statements. The first lemma states that a sufficiently large uniformly random subset of the requests will be accepted in step 2 of each phase.

**Lemma 8.1.** *Denote by $\mathcal{R}$ the set of requests $\mathcal{A}_b$ generates in step 1 of a phase. Provided that $|\mathcal{R}| \in [\omega(\log n), n]$ and $n$ is sufficiently large, w.h.p. a uniformly random subset of $\mathcal{R}$ of size at least $|\mathcal{R}|/4$ is accepted in step 2.*

*Proof.* Set $\lambda := |\mathcal{R}|/n \leq 1$. Consider the random experiment where $|\mathcal{R}| = \lambda n$ balls are thrown u.i.r. into $n$ bins. We make a case differentiation. Assume first that $\lambda \in [1/4, 1]$. Denote for $l \in \mathbb{N}_0$ by $B_l$ the random variable counting the number of bins receiving exactly $l$ balls. According to Corollary 2.16,

$$\begin{aligned} B_1 &\geq \lambda n - 2\left(B_0 - (1-\lambda)n\right) \\ &\in \left(2 - \lambda - 2(1+o(1))e^{-\lambda}\right) n \\ &= \frac{2 - \lambda - 2(1+o(1))e^{-\lambda}}{\lambda}|\mathcal{R}| \end{aligned}$$

w.h.p. Since $\lambda \geq 1/4$, the $o(1)$-term is asymptotically negligible. Without that term, the prefactor is minimized at $\lambda = 1$, where it is strictly larger than $1/4$.

On the other hand, if $\lambda < 1/4$, we may w.l.o.g. think of the balls as being thrown sequentially. In this case, the number of balls thrown into occupied bins is dominated by the sum of $|\mathcal{R}|$ independent Bernoulli variables taking the value 1 with probability $1/4$. Since $|\mathcal{R}| \in \omega(\log n)$, Corollary 2.13 yields that w.h.p. at most $(1/4 + o(1))|\mathcal{R}|$ balls hit non-empty bins. For sufficiently large $n$, we get that w.h.p. more than $(1/2 - o(1))|\mathcal{R}| > |\mathcal{R}|/4$ bins receive exactly one ball.

Finally, consider step 1 of the algorithm. The above considerations show that w.h.p. at least $|\mathcal{R}|/4$ bins receive exactly one request, which they will accept in step 2. Consider such a bin receiving exactly one request. Since each ball sends each element of $\mathcal{R}$ it holds with probability $1/n$ to this bin, the accepted request is drawn uniformly at random from $\mathcal{R}$. Furthermore, as we know that no other copy is sent to the bin, all other messages are sent with conditional probability $1/(n-1)$ each to any of the other bins. Repeating this argument inductively for all bins receiving exactly one request, the claim follows. □

In order to apply the previous lemma repeatedly, we need to make sure that the total number of requests in each round remains smaller than $n$. This is connected to the fact that the fraction of non-placed balls drops exponentially in the number of requests per ball w.h.p.

**Lemma 8.2.** *For a phase $i \in \mathbb{N}$ of $\mathcal{A}_b$, suppose the number of balls that have not been placed yet is bounded by $\beta_i \leq n/k(i)$ w.h.p., where $\beta_i \in \mathbb{R}^+$. Then the number of unplaced balls remaining after phase $i$ is bounded by*

$$\max\left\{(1+o(1))e^{-k(i)/4}\beta_i, e^{-\sqrt{\log n}}n\right\}$$

*w.h.p.*

*Proof.* If $\beta_i \leq e^{-\sqrt{\log n}}n$ or $n$ is constantly bounded, the statement is trivial. Thus, as the $\beta_i$ remaining balls send $k(i)\beta_i \leq n$ requests in phase $i$, Lemma 8.1 states that w.h.p. a uniformly random subset of size at least

$k(i)\beta_i/4$ of the requests is accepted. For each ball, exactly $k(i)$ requests are sent. Hence, if we draw requests one by one, each message that we have not seen yet has probability at least $k(i)/(k(i)\beta_i) = 1/\beta_i$ to occur in the next trial.

Thus, the random experiment where in each step we draw one of the $\beta_i$ balls u.i.r. with probability $1/\beta_i$ each and count the number of distinct balls stochastically dominates the experiment counting the number of placed balls. The former is exactly the balls-into-bins scenario from Lemma 2.15, where (at least) $k(i)\beta_i/4$ balls are thrown into $\beta_i$ bins. For $n$ sufficiently large, we have that

$$k(i) \leq \sqrt{\log n} \leq \frac{2(\ln n - \sqrt{\log n})}{\ln \ln n} \leq \frac{2\ln(k(i)\beta_i)}{\ln \ln n}.$$

Hence, Corollary 2.16 bounds the number of balls that cannot be placed in phase $i$ by $(1+o(1))e^{-k(i)/4}\beta_i$ w.h.p. □

Finally, we need to show that the algorithm places a small number of remaining balls quickly.

**Lemma 8.3.** *Suppose that at the beginning of phase $i_0 \in \mathbb{N}$ of $\mathcal{A}_b$ merely $e^{-\sqrt{\log n}}n$ balls have not been placed yet and $k(i_0) = \sqrt{\log n}$. Then all balls are placed within $\mathcal{O}(1)$ more phases of $\mathcal{A}_b$ w.h.p.*

*Proof.* Regardless of all other balls' messages, each request has probability at least $1 - \sqrt{\log n}\, e^{-\sqrt{\log n}}$ to be accepted. Thus, the probability that a ball cannot be placed is independently bounded by

$$\left(\sqrt{\log n}\, e^{-\sqrt{\log n}}\right)^{k(i_0)} \subseteq n^{-\Omega(1)}.$$

Since $k(i) = k(i_0) = \sqrt{\log n}$ for all $i \geq i_0$, all balls are placed within $\mathcal{O}(1)$ phases w.h.p. □

Plugging these three lemmas together, we derive our performance bounds on $\mathcal{A}_b$.

**Theorem 8.4.** *$\mathcal{A}_b$ solves Problem 6.1, guaranteeing the following properties:*

- *It terminates after* $\log^* n + \mathcal{O}(1)$ *rounds w.h.p.*
- *Each bin in the end contains at most* $\log^* n + \mathcal{O}(1)$ *balls w.h.p.*
- *In each round, the total number of messages sent is at most $n$ w.h.p. The total number of messages is in $\mathcal{O}(n)$ w.h.p.*
- *Balls send and receive $\mathcal{O}(1)$ messages in expectation and at most $\mathcal{O}(\sqrt{\log n})$ messages w.h.p.*

- *Bins send and receive $\mathcal{O}(1)$ messages in expectation and at most $\mathcal{O}(\log n/\log\log n)$ messages w.h.p.*

*Furthermore, the algorithm runs asynchronously in the sense that balls and bins can decide on any request respectively permission immediately, provided that balls' messages contain round counters. According to the previous statements messages then have a size of $\mathcal{O}(1)$ in expectation and $\mathcal{O}(\log\log^* n)$ w.h.p.*

*Proof Sketch (see [68] for technical details).* We apply Lemma 8.2 inductively to see that after a constant number of phases, the number of remaining balls starts to drop exponentially in $k(i)$ in each phase $i$ w.h.p. Moreover, the definition of $k(i)$ and the lemma yield that the total number of messages sent in each phase is monotonically decreasing and falls for sufficiently large $k(i) < \sqrt{\log n}$ exponentially w.h.p. Lemma 8.3 shows that as soon as $k(i)$ becomes $\sqrt{\log n}$, the algorithm terminates in a constant number of rounds w.h.p.

By basic calculations with $\log^*$, one infers that w.h.p. the algorithm terminates within $\log^* n+\mathcal{O}(1)$ phases and thus also rounds, which immediately implies a maximal bin load of $\log^* n + \mathcal{O}(1)$. Together with the above statements about the number of messages sent, this also proves the third and fourth statement as well as the bound on the expected number of messages bins send and receive. The fact that w.h.p. $\mathcal{O}(n)$ messages are sent in total to u.i.r. bins implies by Corollary 2.13 the bound on the maximal number of messages bins send and receive. Since Lemma 8.1 argues about the bins receiving exactly one request in a given phase only, all results also hold for the asynchronous case. $\square$

We remark that a more careful analysis would allow for a smaller cap on the maximal number of requests a ball sends in a given round. This can be seen in the proof of Lemma 8.3, where we did not exploit that the number of remaining balls still drops quickly once the arbitrarily chosen threshold of $e^{-\sqrt{\log n}}n$ is reached, i.e., the probability for a collision falls further. Incurring a larger time complexity of e.g. $(1 + o(1))\log^* n$ or $\mathcal{O}(\log^* n)$ permitted to reduce this bound even more.

The simple approach that motivated Algorithm $\mathcal{A}_b$ is quite flexible, as a number of corollaries will demonstrate. We give only the key arguments of the proofs and refer to [68] for details. Our first observation is that, without surprise, starting with less balls leads to earlier termination.

**Corollary 8.5.** *If only $m := n/\log^{(r)} n$ balls are to be placed into $n$ bins for some $r \in \mathbb{N}$, $\mathcal{A}_b$ initialized with $k(1) := \lfloor \log^{(r)} n\rfloor$ terminates within $r + \mathcal{O}(1)$ rounds w.h.p.*

*Proof.* This can be viewed as the algorithm being started in a later round, and only $\log^* n - \log^*(\log^{(r)} n) + \mathcal{O}(1) = r + \mathcal{O}(1)$ more rounds are required for the algorithm to terminate. □

More interestingly, this can be used to enforce a constant time complexity at the expense of slightly larger bin loads.

**Corollary 8.6.** *For any $r \in \mathbb{N}$, $\mathcal{A}_b$ can be modified into an Algorithm $\mathcal{A}_b(r)$ that guarantees a maximal bin load of $\log^{(r)} n / \log^{(r+1)} n + r + \mathcal{O}(1)$ w.h.p. and terminates within $r + \mathcal{O}(1)$ rounds w.h.p. Its message complexity respects the same bounds as the one of $\mathcal{A}_b$.*

*Proof sketch.* In order to speed up the process, we rule that in the first phase bins accept up to $l := \lfloor \log^{(r)} n / \log^{(r+1)} n \rfloor$ many balls. Computation and Chernoff's bound show that w.h.p. no more than $\operatorname{polylog} n + 2^{-\Omega(l \log l)} n$ balls remain after the first phase. By Corollary 8.5 this implies the statement. □

Another appealing side effect of the adaptive increase in the number of requests is that it deals with faults implicitly.

**Corollary 8.7.** *Algorithm $\mathcal{A}_b$ can be modified to tolerate independent message loss with constant probability $p$. The properties from Theorem 8.4 remain untouched except that balls now send w.h.p. at most $\mathcal{O}(\log n)$ messages.*

*Proof Sketch.* In step 4 of $\mathcal{A}_b$ we set

$$k(i+1) := \min\{k(i) e^{\lfloor (1-p)^2 k(i) \rfloor / 5}, \log n\}.$$

Essentially the reasoning remains the same, except that a request or the respective response may get lost with independent probability $(1-p)$ each, necessitating to increase $k(i)$ more slowly. Moreover, even if few balls remain, still each request has constant probability to fail. Therefore, we need to increase the bound on $k(i)$ to $\log n$ to ensure quick termination. □

The same technique permits to enforce a maximal bin load of two.

**Corollary 8.8.** *We modify $\mathcal{A}_b$ into $\mathcal{A}_b^2$ by ruling that any bins having already accepted two balls refuse any further requests in step 2, and in step 4 we set*

$$k(i+1) := \min\{k(i) e^{\lfloor k(i) \rfloor / 10}, \log n\}.$$

*Then the statements of Theorem 8.4 remain true except that balls now send $\mathcal{O}(\log n)$ instead of $\mathcal{O}(\sqrt{\log n})$ messages w.h.p. In turn, the maximal bin load of the algorithm becomes two.*

*Proof Sketch.* Trivially, at any time no more than half of the bins may have two balls placed into them. Thus, if balls inform the bins they commit to, always at least half of the bins are capable of accepting a ball. Consequently, the same reasoning as for Corollary 8.7 applies. □

The observation that neither balls nor bins need to wait prior to reacting to a message implies that our algorithms can also be executed sequentially, placing one ball after another. In particular, we can guarantee a bin load of two efficiently. This corresponds to the simple sequential algorithm that queries for each ball sufficiently many bins to find one of load less than two.

**Lemma 8.9.** *An adaptive sequential balls-into-bins algorithm $\mathcal{A}_{seq}$ exists guaranteeing a maximal bin load of two, requiring at most $(2+o(1))n$ random choices and bin queries w.h.p.*

*Proof.* The algorithm simply queries u.i.r. bins until one of load less than two is found; then the current ball is placed and the algorithm proceeds with the next. Since at least half of the bins have load less than two at any time, each query has independently a probability of at least $1/2$ of being successful. Therefore, it can be deduced from Corollary 2.13 that no more than $(2 + o(1))n$ bin queries are necessary to place all balls w.h.p. □

## 8.2 Optimal Asymmetric Algorithm

In this section, we will show that asymmetric algorithms can indeed obtain constant bin loads in constant time, at asymptotically optimal communication costs. Note that for asymmetric algorithms, we can w.l.o.g. assume that $n$ is a multiple of some number $l \in o(n)$, since we may simply opt for ignoring negligible $n - l\lfloor n/l \rfloor$ bins. We start by presenting a simple algorithm demonstrating the basic idea of our solution. Given $l \in \mathcal{O}(\log n)$ that is a factor of $n$, $\mathcal{A}_1(l)$ is defined as follows.

1. Each ball contacts one bin chosen uniformly at random from the set $\{il \,|\, i \in \{1, \dots, n/l\}\}$.
2. Bin $il$, $i \in \{1, \dots, n/l\}$, assigns up to $3l$ balls to the bins $(i-1)l + 1, \dots, il$, such that each bin gets at most three balls.
3. The remaining balls (and the bins) proceed as if executing the symmetric Algorithm $\mathcal{A}_b^2$, however, with $k$ initialized to $k(1) := 2^{\alpha l}$ for an appropriately chosen constant $\alpha > 0$.

Essentially, we create buckets of non-constant size $l$ in order to ensure that the load of these buckets is slightly better balanced than it would be the case for individual bins. This enables the algorithm to place more than a constant fraction of the balls immediately. Small values of $l$ suffice for this algorithm to terminate quickly.

**Lemma 8.10.** *Algorithm $\mathcal{A}_1(l)$ solves Problem 6.4 with a maximal bin load of three. It terminates within $\log^* n - \log^* l + \mathcal{O}(1)$ rounds w.h.p.*

*Proof.* Let for $i \in \mathbb{N}_0$ $Y^i$ denote the random variables counting the number of bins receiving at least $i$ messages in step 1. From Lemma 2.15 we know that Corollary 2.13 applies to these variables, i.e., $|Y^i - \mathbb{E}[Y^i]| \in \mathcal{O}\left(\log n + \sqrt{\mathbb{E}[Y^i]\log n}\right)$ w.h.p. Consequently, we have that the number $Y^i - Y^{i+1}$ of bins receiving exactly $i$ messages differs by at most $\mathcal{O}\Big(\log n + \sqrt{\max\{\mathbb{E}[Y^i], \mathbb{E}[Y^{i+1}]\}\log n}\Big)$ from its expectation w.h.p. Moreover, Corollary 2.13 states that these bins receive at most $l + \mathcal{O}(\sqrt{l \log n} + \log n) \subset \mathcal{O}(\log n)$ messages w.h.p., i.e., we need to consider values of $i \in \mathcal{O}(\log n)$ only.

Thus, the number of balls that are not accepted in the first phase is bounded by

$$\begin{aligned}
&\sum_{i=3l+1}^{n} (i-3l)\left(Y^i - Y^{i+1}\right)\\
\in\ &\sum_{i=3l+1}^{\mathcal{O}(\log n)} (i-3l)\,\mathbb{E}\Big[Y^i - Y^{i+1}\Big] + \mathcal{O}\left(\sqrt{n\log n}\right)\\
\subseteq\ &\frac{n}{l}\sum_{i=3l+1}^{\mathcal{O}(\log n)} (i-3l)\binom{n}{i}\left(\frac{l}{n}\right)^i\left(1-\frac{l}{n}\right)^{n-i} + \mathcal{O}\left(\sqrt{n\log n}\right)\\
\subseteq\ &\frac{n}{l}\sum_{i=3l+1}^{\mathcal{O}(\log n)} (i-3l)\left(\frac{el}{i}\right)^i + \mathcal{O}\left(\sqrt{n\log n}\right)\\
\subseteq\ &\frac{n}{l}\sum_{j=1}^{\infty} jl\left(\frac{e}{3}\right)^{(j+2)l} + \mathcal{O}\left(\sqrt{n\log n}\right)\\
\subseteq\ &\mathcal{O}\left(\left(\frac{e}{3}\right)^{2l} n + \sqrt{n\log n}\right)\\
\subseteq\ &\left(\frac{3}{e}\right)^{-(2-o(1))l} n + \mathcal{O}\left(\sqrt{n\log n}\right)
\end{aligned}$$

w.h.p., where in the third step we used the inequality $\binom{n}{i} \leq (en/i)^i$.

Thus, w.h.p. at most $2^{-\Omega(l)}n + \mathcal{O}\left(\sqrt{n\log n}\right)$ balls are not assigned in the first two steps. Hence, according to Corollary 8.5, we can deal with the remaining balls within $\log^* n - \log^* l + \mathcal{O}(1)$ rounds by running $\mathcal{A}_b^2$ with $k$ initialized to $2^{\alpha l}$ for $\alpha \in (2 - o(1)) \log(3/e)$ when executing $\mathcal{A}_s$. We conclude that $\mathcal{A}_1(l)$ will terminate after $\log^* n - \log^* l + \mathcal{O}(1)$ rounds w.h.p. as claimed. □

In particular, if we set $l := \log^{(r)} n$, for any $r \in \mathbb{N}$, the algorithm terminates within $r + \mathcal{O}(1)$ rounds w.h.p. However, the result of Lemma 8.10 is somewhat unsatisfactory with respect to the balls-into-bins problem, since a subset of the bins has to deal with an expected communication load of $l + \mathcal{O}(1) \in \omega(1)$. Hence, we want to modify the algorithm such that this expectation is constant.

To this end, assume that $l \in \mathcal{O}(\log n / \log\log n)$ and $l^2$ is a factor of $n$. Consider the following algorithm $\mathcal{A}_2(l)$ which assigns balls as *coordinators* of intervals of up to $l^2$ consecutive bins.

1. With probability $1/l$, each ball picks one bin interval $I_j := \{(j-1)l + 1, \ldots, jl\}$, $j \in \{1, \ldots, n/l\}$, uniformly at random and contacts these bins. These messages from each ball contain a (for each ball fixed) string of $\lceil (c+2) \log n \rceil$ random bits.

2. Each bin that receives one or more messages sends an acknowledgement to the ball whose random string represents the smallest number; if two or more strings are identical, no response is sent.

3. Each ball $b$ that received acknowledgements from a contacted interval $I_j$ queries one u.i.r. chosen bin from each interval $I_{j+1}, \ldots, I_{j+l-1}$ (taking indices modulo $n/l$) whether it has previously acknowledged a message from another ball; these bins respond accordingly. Ball $b$ becomes the coordinator of $I_j$ and all consecutive intervals $I_{j+1}, \ldots, I_{j+k}$, $k < l$ such that none of these intervals has already responded to another ball in step 2.

The algorithm might miss some bin intervals, but overall most of the bins will be covered.

**Lemma 8.11.** *When $\mathcal{A}_2(l)$ terminates after a constant number of rounds, w.h.p. all but $2^{-\Omega(l)} n$ bins have a coordinator. The number of messages sent or received by each ball is constant in expectation and at most $\mathcal{O}(l)$. The number of messages sent or received by each bin is constant in expectation and $\mathcal{O}(\log n / \log\log n)$ w.h.p. The total number of messages is in $\mathcal{O}(n)$ w.h.p.*

*Proof.* The random strings chosen in step 2 are unique w.h.p., since the probability that two individual strings are identical is at most $2^{-(c+2)\log n} = n^{-(c+2)}$ and we have $\binom{n}{2} < n^2$ different pairs of balls. Hence, we may w.l.o.g.

assume that no identical strings are received by any bins in step 2 of the algorithm.

In this case, if for some $j$ the bins in $I_j$ are not contacted in steps 1 or 3, this means that $l$ consecutive intervals were not contacted by any ball. The probability for this event is bounded by

$$\left(1-\frac{1}{l}\cdot\frac{l}{n/l}\right)^n=\left(1-\frac{l}{n}\right)^n<e^{-l},$$

Hence, in expectation less than $e^{-l}n/l$ intervals get no coordinator assigned.

The variables indicating whether the $I_j$ have a coordinator are negatively associated, as can be seen as follows. We interpret the first round as throwing $n$ balls u.i.r. into $n$ bins, of which $n/l$ are labeled $I_1,\ldots,I_{n/l}$. An interval $I_j$ has a coordinator exactly if one of the bins $I_{j-l+1},\ldots,I_j$ (again, indices modulo $n/l$) receives a ball. We know from Lemma 2.15 that the indicator variables $Y_i^1$ counting the non-empty bins are negatively associated; however, the third step of its proof uses Statement (*iii*) from Lemma 2.14, which applies to *any* set of increasing functions. Since maxima of increasing functions are increasing, also the indicator variables $\max\{Y^1_{I_{j-l+1}},Y^1_{I_{j-l+2}},\ldots,Y^1_{I_j}\}$ are negatively associated.

Therefore, Corollary 2.13 yields that the number of intervals that have no coordinator is upper bounded by $\mathcal{O}(e^{-l}n/l+\log n)$ w.h.p. Consequently, w.h.p. all but $\mathcal{O}(e^{-l}n+l\log n)\subseteq e^{-\Omega(l)}n$ bins are assigned a coordinator.

Regarding the communication complexity, observe that balls send at most $\mathcal{O}(l)$ messages and participate in the communication process with probability $1/l$. In expectation, $n/l$ balls contact u.i.r. chosen bin intervals, implying that bins receive in expectation one message in step 1. Similarly, at most $l$ balls pick u.i.r. bins from each $I_j$ to contact in step 3. Since in step 3 at most $l-1$ messages can be received by bins, it only remains to show that the bound of $\mathcal{O}(\log n/\log\log n)$ on the number of messages bins receive in step 1 holds. This follows from the previous observation that we can see step 1 as throwing $n$ balls u.i.r. into $n$ bins, where $n/l$ bins represent the $I_j$. For this setting the bound follows from Lemma 2.15 and Corollary 2.13. Finally, we apply Corollary 2.13 to the number of balls choosing to contact bins in step 1 in order to see that $\mathcal{O}(n)$ messages are sent in total w.h.p. □

Finally, algorithm $\mathcal{A}(l)$ essentially plugs $\mathcal{A}_1(l)$ and $\mathcal{A}_2(l)$ together, where $l\in\mathcal{O}(\sqrt{\log n})$ and $l^2$ is a factor of $n$.

1. Run Algorithm $\mathcal{A}_2(l)$.

2. Each ball contacts one bin, chosen uniformly.

3. Each coordinator contacts the bins it has been assigned to by $\mathcal{A}_2(l)$.

4. The bins respond with the number of balls they received a message from in step 2.

5. The coordinators assign (up to) three of these balls to each of their assigned bins. They inform each bin where the balls they received messages from in step 2 need to be redirected.

6. Each ball contacts the same bin as in step 2. If the bin has a coordinator and the ball has been assigned to a bin, the bin responds accordingly.

7. Any ball receiving a response informs the respective bin that it is placed into it and terminates.

8. The remaining balls (and the bins) proceed as if executing Algorithm $\mathcal{A}_b^2$, however, with $k$ initialized to $k(1) := 2^{\alpha l}$ for an appropriately chosen constant $\alpha > 0$.

**Theorem 8.12.** *Algorithm $\mathcal{A}(l)$ solves Problem 6.4 with a maximal bin load of three. It terminates after $\log^* n - \log^* l + \mathcal{O}(1)$ rounds w.h.p. Both balls and bins send and receive a constant number of messages in expectation. Balls send and receive at most $\mathcal{O}(\log n)$ messages w.h.p., bins $\mathcal{O}(\log n / \log\log n)$ many w.h.p. The total number of messages is in $\mathcal{O}(n)$ w.h.p.*

*Proof.* Lemma 8.11 states that all but $2^{-\Omega(l)}n$ bins have a coordinator. Steps 2 to 7 of $\mathcal{A}(l)$ emulate steps 1 and 2 of Algorithm $\mathcal{A}_1(l)$ for all balls that contact bins having a coordinator. By Lemma 8.10, w.h.p. all but $2^{-\Omega(l)}n$ of the balls could be assigned if the algorithm would be run completely, i.e., with all bins having a coordinator. Since w.h.p. only $2^{-\Omega(l)}n$ bins have no coordinator and bins accept at most three balls, we conclude that w.h.p. after step 7 of $\mathcal{A}(l)$ merely $2^{-\Omega(l)}n$ balls have not been placed into bins. Thus, analogously to Lemma 8.10, step 8 will require at most $\log^* n - \log^* l + \mathcal{O}(1)$ rounds w.h.p. Since steps 1 to 7 require constant time, the claimed bound on the running time follows.

The bounds on message complexity and maximal bin load are direct consequences of Corollary 8.5, Lemma 8.11, the definition of $\mathcal{A}(l)$, and the bound of $\mathcal{O}(\sqrt{\log n})$ on $l$. □

Thus, choosing $l = \log^{(r)} n$ for any $r \in \mathbb{N}$, Problem 6.4 can be solved within $r + \mathcal{O}(1)$ rounds.

## 8.3 Symmetric Solution Using $\omega(n)$ Messages

A similar approach is feasible for symmetric algorithms if we permit $\omega(n)$ messages in total. Basically, Algorithm $\mathcal{A}(l)$ relied on asymmetry to assign coordinators to a vast majority of the bins. Instead, we may settle for coordinating a constant fraction of the bins; in turn, balls will need to send $\omega(1)$ messages to find a coordinated bin with probability $1 - o(1)$.

Consider the following Algorithm $\mathcal{A}_\omega(l)$, where $l \leq n/\log n$ is integer.

1. With probability $n/l$, a ball contacts a uniformly random subset of $l$ bins.

2. Each bin receiving at least one message responds to one of these messages, choosing arbitrarily. The respective ball is the coordinator of the bin.

This simple algorithm guarantees that a constant fraction of the bins will be assigned to coordinators of $\Omega(l)$ bins.

**Lemma 8.13.** *When executing $\mathcal{A}_\omega(l)$, bins receive $\mathcal{O}(\log n/\log\log n)$ messages w.h.p. In total $\mathcal{O}(n)$ messages are sent w.h.p. W.h.p., a constant fraction of the bins is assigned to coordinators of $\Omega(l)$ bins.*

*Proof.* Corollary 2.13 states that in step 1 w.h.p. $\Theta(n/l)$ balls decide to contact bins, i.e., $\Theta(n)$ messages are sent. As before, the bound on the number of messages bins receive follows from Lemma 2.15 and Corollary 2.13. Using Lemma 2.15 and Corollary 2.13 again, we infer that w.h.p. a constant fraction of the bins receives at least one message. Thus, $\Theta(n/l)$ balls coordinate $\Theta(n)$ bins, implying that also $\Theta(n)$ bins must be coordinated by balls that are responsible for $\Omega(l)$ bins. □

Permitting communication exceeding $n$ messages by more than a constant factor, this result can be combined with the technique from Section 8.1 to obtain a constant-time symmetric algorithm.

**Corollary 8.14.** *For $l \in \mathcal{O}(\log n)$, an Algorithm $\mathcal{A}_c(l)$ exists that sends $\mathcal{O}(ln)$ messages w.h.p. and solves Problem 6.3 with a maximal bin load of $\mathcal{O}(1)$ within $\log^* n - \log^* l + \mathcal{O}(1)$ rounds w.h.p. Balls send and receive $\mathcal{O}(l)$ messages in expectation and $\mathcal{O}(\log n)$ messages w.h.p.*

*Proof Sketch.* W.h.p., Algorithm $\mathcal{A}_\omega(l)$ assigns coordinators to a constant fraction of the bins such that these coordinators control $l_0 \in \Omega(l)$ bins. The coordinators inform each of their bins $b$ of the number of bins $\ell(b)$ they supervise, while any other ball contacts a uniformly random subset of $l$ bins. Such a bin $b$, if it has a coordinator, responds with the value $\ell(b)$. Note that the probability that the maximal value a ball receives is smaller than $l_0$ is smaller than $2^{-\Omega(l)}$; Corollary 2.13 therefore states that w.h.p. $(1-2^{-\Omega(l)})n$ balls contact a bin $b$ with $\ell(b) \geq l_0$.

Next, these balls contact a bin $b$ from which they received a value $\ell(b) \geq l_0$, where they pick a feasible bin uniformly at random. The respective coordinators assign (at most) constantly many of these balls to each of their bins. By the same reasoning as in Lemma 8.10, we see that (if the constant was sufficiently large) all but $2^{-\Omega(l)}n$ balls can be placed. Afterwards, we again proceed as in Algorithm $\mathcal{A}_b^2$, with $k$ initialized to $2^{\alpha l}$ for an appropriate $\alpha > 0$; analogously to Lemma 8.10 we obtain the claimed running bound. The bounds on message complexity can be deduced from Chernoff bounds as usual. □

Again, choosing $l = \log^{(r)} n$ for any $r \in \mathbb{N}$, Problem 6.3 can be solved within $r + \mathcal{O}(1)$ rounds using $\mathcal{O}(n \log^{(r)} n)$ messages w.h.p.

## 8.4 An Application

We conclude this chapter by briefly presenting an exemplary application of our results. Consider the following routing problem. Assume we have a system of $n$ fully connected nodes, where links have uniform capacity. All nodes have unique identifiers, that is, $v \in V$ denotes both the node $v$ and its identifier. For the sake of simplicity, let communication be synchronous and reliable. During each synchronous round, nodes may perform arbitrary local computations, send a (different) message to each other node, and receive messages.

Ideally, we would like to fully exploit the outgoing and incoming bandwidth (whichever is more restrictive) of each node with marginal overhead w.h.p. More precisely, we strive for enabling nodes to freely divide the messages they can send in each round between all possible destinations in the network. Naturally, this is only possible to the extent dictated by the capability of nodes to receive messages in each round, i.e., the amount of time required can at best be proportional to the maximal number of messages any node must send or receive, divided by $n$.

This leads to the following problem formulation.

**Problem 8.15** (Information Distribution Task). *Each node $v \in V$ is given a (finite) set of messages*

$$\mathcal{S}_v = \{m_v^i \mid i \in I_v\}$$

*with destinations $d(m_v^i) \in V$, $i \in I_v$. Messages can be distinguished (e.g., by including the sender's identifier and the position in an internal ordering* *of the messages of that sender). The goal is to deliver all messages to their destinations, minimizing the total number of rounds. By*

$$\mathcal{R}_v := \left\{ m_w^i \in \bigcup_{w \in V} \mathcal{S}_w \,\middle|\, d(m_w^i) = v \right\}$$

*we denote the set of messages a node $v \in V$ shall receive. We abbreviate $M_s := \max_{v \in V} |\mathcal{S}_v|$ and $M_r := \max_{v \in V} |\mathcal{R}_v|$, i.e., the maximal numbers of messages a single node needs to send or receive, respectively.*

Note that this task is not trivial, as nodes have no information on who wants to send messages to whom, while indirection is crucial to achieve a small time complexity. Abstractly speaking, we have $n$ concurrent balls-into-bins problems: If we can for each node $v \in V$ distribute the messages destined to it among all other nodes in a constant number of rounds, such that no node holds more than constantly many messages for each destination, all messages can be delivered within constant time. Thus, using the approach from Section 8.2, the following statement can be derived (cf. [68]).

**Theorem 8.16.** *Problem 8.15 can be solved in*

$$\mathcal{O}\left(\frac{M_s + M_r}{n}\right)$$

*rounds w.h.p.*

As can be seen from this bound, it is feasible to apply our techniques also if the number of balls $m$ is not at most the number of bins $n$. Essentially, this is accounted for by increasing the maximal bin load to $\mathcal{O}(m/n)$.

Similar results are obtained if one bases a solution on Algorithm $\mathcal{A}_b$, however, with an additive term of $\mathcal{O}(\log^* n)$ in the time needed to solve Problem 8.15 [68]. Figure 8.1 shows simulation results agreeing with the bounds from our analysis.

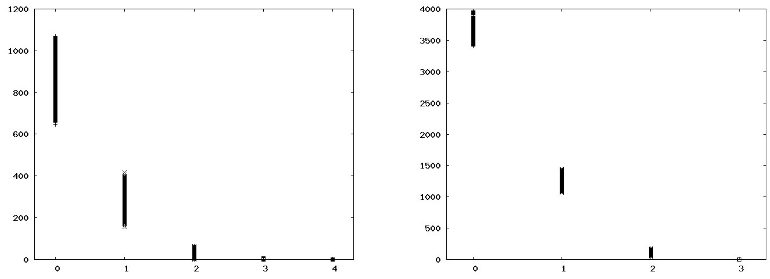

Figure 8.1: Simulations of a solution to Problem 8.15 based on Algorithm $\mathcal{A}_b$. Number of remaining messages plotted against passed phases (taking two rounds each). 512 runs were simulated for 32 (left) and 64 (right) nodes. $(M_s + M_r)/n$ was close to 2 for all runs. In both cases, most of the instances terminated after three phases; for 32 nodes, a single instance required 5 phases.

# Part III

# Graph Problems in Restricted Families of Graphs

# Chapter 9

# An Introduction to Graph Problems

*"How many papers have you published yet?" – "Three." – "And how long are your studies supposed to take?" – "Three years." – "Then you have to write six more papers before you're done!" – My mother's approximation to the number of publications that make up a thesis.*

In large parts, theoretical computer science is the study of the complexity of archetypical problems, many of which are naturally formulated on graphs. Traditionally, one primarily examines how many sequential computing steps are required to solve an instance of a problem in the worst case. This is measured in terms of the input size, frequently expressed by the number of nodes $n$. With the advent of distributed computing, this approach had to be reconsidered. While NP-hard problems remain (supposedly) intractable even if one increases the computing power by factor $n$, the main hurdles for "solvable" graph problems in distributed systems are usually limits on communication and concurrency. In fact, classical distributed symmetry breaking problems like finding a coloring of the nodes with $\Delta + 1$ colors (see Definition 2.25) or a maximal set of non-adjacent nodes (see Definition 9.6 are completely trivial from the perspective of sequential algorithms.

In this part of this thesis, we consider two well-known graph problems, the first being the minimum dominating set problem.

**Definition 9.1** (Dominating Sets)**.** *Given a graph $G = (V, E)$, a node $v \in V$ (set $A \subseteq V$)* covers *$\mathcal{N}_v^+$ ($\mathcal{N}_A^+$). The set $D \subseteq V$ is a* dominating set (DS) *if it covers $V$. A dominating set of minimal cardinality is a* minimum dominating set (MDS)*.*

As easy as it is to state the problem of finding a minimum dominating set, as difficult it is to solve. In fact, finding a minimum dominating set was one of the first tasks known to be NP-hard [37]. Consequently, one typically is satisfied with an approximative solution.

**Definition 9.2** (Minimum Dominating Set Approximations)**.** *Given $f \in \mathbb{R}^+$, a DS $D$ is an $f$* MDS approximation*, if $|D| \leq f|M|$ for any MDS $M$. For $f : \mathbb{N} \to \mathbb{R}^+$, an $f$* approximation algorithm *for the MDS problem returns for any feasible graph of $n$ nodes a DS that is an $f(n)$ approximation. For randomized algorithms this might happen only with at least a certain probability; this probability bound however must not depend on the particular instance.*

In general, it is also NP-hard to compute a $C \log \Delta$ approximation [95], where $C > 0$ is some constant. Indeed, unless NP is a subset of the problems that can be solved within $n^{\mathcal{O}(\log \log n)}$ steps, a $(1 - o(1)) \ln \Delta$ approximation is intractable in general graphs.

While this bound is easily matched by the centralized algorithm that always picks the node covering the most yet uncovered nodes, the distributed case is more involved. Here, the lower bound on the approximation ratio can be asymptotically matched within $\mathcal{O}(\log n)$ rounds by a randomized algorithm [56], whereas any distributed algorithm achieving a polylogarithmic approximation must take $\Omega(\log \Delta)$ and $\Omega(\sqrt{\log n})$ rounds [54]. Therefore, the algorithm from [56] is $\mathcal{O}(\log n / \log \Delta)$-optimal with respect to the running time. We remark, though, that the algorithm makes use of messages whose size respects no non-trivial bound, i.e., nodes might need to learn about the entire graph. For this reason, the authors also propose a variant of their algorithm running in $\mathcal{O}(\log^2 \Delta)$ rounds with message size $\mathcal{O}(\log \Delta)$, yielding a complexity gap of $\mathcal{O}(\log \Delta)$ in running time.

While not tight, these results seem to suggest that we are not far from understanding the distributed complexity of the MDS problem. We believe that this is not the case. The lower bound from [54] is based on graphs that have large girth, yet many edges. Although such graphs exist and show that there is no algorithm solving the problem more efficiently in *any* graph of $n$ nodes or maximum degree $\Delta$, in a practical setting one will never encounter one of these constructed instances. Thus, we argue that it is reasonable to restrict inputs to graph families which occur in realistic settings. Of course, this approach suffers from the drawback that it is not trivial to find appropriate families of graphs—supposing they even exist—offering both sufficiently efficient solutions as well as wide practical applicability. We do not claim to have a satisfying answer to this question. Instead, we confine ourselves to striving for an extended knowledge on the complexity of the MDS problem in restricted families of graphs.

To this end, we devise two MDS approximation algorithms for graphs of small arboricity presented in Chapter 12.

**Definition 9.3** (Forest Decomposition and Arboricity)**.** *For $f \in \mathbb{N}_0$, an $f$-*forest decomposition *of a graph $G = (V, E)$ is a partition of the edge set into $f$ rooted forests. The* arboricity *$A(G)$ is the minimum number of forests in a forest decomposition of $G$.*

The graph class of (constantly) bounded arboricity is quite general, as any family of graphs excluding a fixed minor has bounded arboricity.

**Definition 9.4** (Contractions and Minors)**.** *Given a simple graph $G$, a* minor *of $G$ can be obtained by any sequence of the following operations.*

- *Deleting an edge.*
- *Deleting a node.*
- Contracting *an edge $\{v, w\}$, i.e., replacing $v$ and $w$ by a new node $u$ such that $\mathcal{N}_u := (\mathcal{N}_v \cup \mathcal{N}_w)/\{v, w\}$.*

Note, however, that graphs of bounded arboricity may contain arbitrary minors. In particular, if we take the complete graph $K_{\sqrt{n}}$ of $\sqrt{n}$ nodes and replace its edges by edge-disjoint paths of length two, we obtain a graph of fewer than $n$ nodes and arboricity two that has $K_{\sqrt{n}}$ as minor. Therefore, demanding bounded arboricity is considerably less restrictive than excluding the existence of certain minors.

Both presented algorithms improve on the results from [56] that apply to general graphs. However, the lower bound from [54] does not hold for graphs of bounded arboricity, yet these algorithms have logarithmic running times. In Chapter 13, we present a different approach that on planar graphs achieves an $\mathcal{O}(1)$ approximation in a few number of rounds.

**Definition 9.5** (Planarity)**.** *A graph $G$ is* planar *if and only if it can be drawn in the two-dimensional plane such that no two nodes are mapped to the same point and edges intersect at their endpoints only. Equivalently, $G$ is planar if and only if it does neither contain $K_{3,3}$ nor $K_5$ as a minor, where $K_{k,k}$, $k \in \mathbb{N}$, is the complete bipartite graph of $k$ nodes each and $K_k$ is the complete graph of $k$ nodes.*

Although our algorithm for planar graphs is not practical due to the use of large messages, we deem this result interesting because it shows that a fast solution exists in a graph family where an $\mathcal{O}(1)$ approximation is not immediately evident. In contrast, in trees the set of inner nodes forms a three approximation, for instance, and in graphs of bounded degree $\Delta$ taking all nodes yields a $\Delta + 1$ approximation.

Our algorithms and the one from [56] share two similarities. On the one hand, they all act greedily in one way or the other. Considering that the sequential algorithm matching the $(1-o(1))\ln n$ lower bound on the approximation ratio on general graphs is also greedy, this could be anticipated. More interestingly, a main obstacle for all these algorithms is symmetry breaking. While in [56] this is achieved by randomized rounding, our algorithms exploit the additional structure present in graphs of small arboricity and planar graphs, respectively. Nevertheless, one of our algorithms additionally relies on randomized symmetry breaking, employing a standard technique for constructing a so-called maximal independent set.

**Definition 9.6** (Independent Sets)**.** *Given a graph $G=(V,E)$, a subset of the nodes $I\subseteq V$ is an* independent set (IS)*, if for any $v,w\in I$, $v\neq w$, we have that $\{v,w\}\notin E$. An IS is a* maximal independent set (MIS)*, if no nodes can be added without destroying independence, i.e., for all $v\in V\setminus I$, the set $I\cup\{v\}$ is not independent. A* maximum independent set (MaxIS) *is an IS of maximal size.*

Note that any MIS is a DS. Moreover, an MIS can be very different from a MaxIS, as can be seen by example of the star graph.

Computing an MIS sequentially is trivial, whereas in general graphs the best known distributed solutions run in $\mathcal{O}(\log n)$ randomized rounds [2, 43, 73, 80]. In case of MIS the lower bound construction from [54] applies to line graphs and proves this to be optimal up to a factor of $\mathcal{O}(\sqrt{\log n})$. Except for graph classes where the problem is much simpler because the number of independent nodes in a neighborhood is small, no considerably faster distributed algorithms are known. In particular, even in a (non-rooted) forest the fastest solution takes $\mathcal{O}(\log n/\log\log n)$ rounds, while the strongest lower bound is $\Omega(\log^* n)$ [70, 89]. This startling complexity gap motivates to study the MIS problem in forests, which we do in Chapter 10.

Finally, we consider unit disk graphs, where the problems of finding an MIS and approximating an MDS are closely related, as any MIS is a constant-factor MDS approximation.

**Definition 9.7** (Unit Disk Graphs)**.** *A* unit disk graph *$G(\iota)=(V,E)$ is defined by a mapping $\iota:V\to\mathbb{R}^2$, where $E:=\left\{\{v,w\}\in\binom{V}{2}\mid \|v-w\|_{\mathbb{R}^2}\leq 1\right\}$.*

For this family of graphs, we generalize the lower bound from [70] in Chapter 11 to show that no deterministic algorithm can find a constant-factor MDS approximation in $o(\log^* n)$ rounds; this bound has been matched asymptotically [98].

Unit disk graphs and generalizations thereof have been employed in the study of wireless communication to model interference and communication ranges [53, 76]. The algorithm from [98] is efficient for the much larger class of graphs of *bounded growth* which satisfies that the number of independent nodes up to a certain distance $r$ is bounded by some function $f(r)$ that is independent of $n$. Thus, together these bounds classify the deterministic complexity of finding a constant MDS approximation in geometric graph families up to constants. We remark, though, that neither the technique from [98] nor our lower bound apply if one uses the more sophisticated *physical model* (see e.g. [88] and references therein).

## 9.1 Model

In this part of the thesis, we make use of a very simple network model. We assume a fault-free distributed system. A simple graph $G = (V, E)$ describes the MDS or MIS problem instance, respectively, as well as the communication infrastructure. In each synchronous round, each node $v \in V$ may send a (different) message to each of its neighbors $w \in \mathcal{N}_v$, receives all messages from its neighbors, and may perform arbitrary finite local computations. Initially, node $v$ knows its neighbors $\mathcal{N}_v$ and possibly a unique identifier of size $\mathcal{O}(\log n)$.

For some algorithms a *port numbering* is sufficient, i.e., the node $v$ has a bijective mapping $p(v, \cdot) : \mathcal{N}_v \to \{1, \dots, |\mathcal{N}_v|\}$ at hand. When sending a message, it may specify which neighbor receives which message by means of the respective port numbers. When receiving, it can tell apart which neighbor sent which message, also in terms of its port numbering. At termination, the node must output whether it is part of the DS or MIS, respectively, and these outputs must define a valid solution of the problem with regard to $G$.

In the context of distributed graph algorithms, this abstract model can be motivated as follows.

- Asynchronicity can be dealt with by a synchronizer (cf. [3]).

- Recovery from transient faults can be ensured by making the algorithm *self-stabilizing* (see Definition 5.27). There is a simple transformation from algorithms obeying the given model to self-stabilizing ones [5, 64].

- Changing topology due to joining and leaving nodes, crash failures, etc. also changes the input, i.e., we need to rerun the algorithm on the new topology.

- With respect to lower bounds, we typically restrict neither the number of messages nor their size. For algorithms, these values should of course be small. This is not enforced by the model, but considered as quality measure like the running time.

- Local computation and memory are typically not critical resources, as the most efficient algorithms usually are not highly complicated (within a small number of rounds, only little information is available that can be processed).

Note that if the algorithm terminates within $T$ rounds, recovery from faults or adaption to new topology require *local* operations up to distance at most $T$ from the event only. In particular, if $T$ is sublogarithmic and degrees are bounded, we get a non-trivial bound on the size of the subgraph that may affect the outcome of a node's computations. This underlines the importance of both upper and lower bounds on the time complexity of algorithms in this model; for small time complexities, this might even be the most significant impact of such bounds – for applications the difference between 5 and 10 rounds of communication might be of no concern, but whether a small fraction or the majority of the nodes has to re-execute the algorithm and change its state in face of e.g. a single node joining the network could matter a lot.

## 9.2 Overview

In Chapter 10 we propose an algorithm computing an MIS in non-rooted forests. For rooted forests, the problem is known to have a time complexity of $\Theta(\log^* n)$ [19, 89]. In contrast, for non-rooted forests no stronger lower bound is known, yet the previously fastest algorithm requires $\Theta(\log n/\log\log n)$ rounds [9]. With a running time of $\mathcal{O}(\sqrt{\log n \log\log n})$, our result reduces this gap considerably.

To be fair, the deterministic algorithm from [9] is efficient for the much more general class of graphs of bounded arboricity. However, it makes use of an $f$-forest decomposition requiring at least $\Omega(\log n/\log f)$ rounds, as derived in [9] from a coloring lower bound by Linial [70]. Hence, this technique cannot be made faster. Moreover, once the decomposition is obtained, the respective algorithm makes heavy use of the fact that the outdegree (i.e., the number of parents) of each node is small. Similarly, other fast solutions that run in time $\Theta(\log^* n)$ on rooted forests [19], graphs of bounded degree [40], or graphs of bounded growth [98] exploit that in one way or another the number of neighbors considered by each node can be kept small. By partly related techniques, one can color the graph with $\Delta+1$ colors and subsequently incrementally add still independent nodes of each color to an IS concurrently, until eventually an MIS has been constructed. This results in deterministic algorithms with a running time of $\mathcal{O}(\Delta + \log^* n)$ [8, 49]. See Table 9.1 for an overview of results on MIS computation.

In light of these results and the fact that forests are a very restrictive graph family, maybe the most interesting point about the algorithm presented

Table 9.1: Upper and lower bounds on distributed time complexities of the MIS problem in various graph families.

| graph family | running time | deterministic |
|---|---|---|
| general [2, 43, 73, 80] | $\mathcal{O}(\log n)$ | no |
| general [90] | $2^{\mathcal{O}(\sqrt{\log n})}$ | yes |
| line graphs [54] | $\Omega\left(\min\left\{\sqrt{\log n}, \log \Delta\right\}\right)$ | no |
| rings & forests [73, 89] | $\frac{\log^* n}{2} - \mathcal{O}(1)$ | no |
| rings & rooted forests [19] | $\mathcal{O}(\log^* n)$ | yes |
| maximum degree $\Delta$ [8, 49] | $\mathcal{O}(\Delta + \log^* n)$ | yes |
| bounded growth [98] | $\mathcal{O}(\log^* n)$ | yes |
| bounded arboricity [9] | $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ | yes |
| forests (Chapter 10) | $\mathcal{O}\left(\sqrt{\log n \log \log n}\right)$ | no |

in Chapter 10 is the following: Despite an unbounded number of independent neighbors and the lack of a conducive edge orientation, it can break symmetry in considerably sub-logarithmic time.

In Chapter 11, we turn to studying the problem of approximating an MDS in unit disk graphs. Leveraging Linial's lower bound [70] of $(\log^* n - 1)/2$ on the number of rounds required to compute a 3-coloring or maximal independent set on the ring, we can show that no deterministic algorithm can compute an $f(n)$ approximation in $g(n)$ rounds if $f(n)g(n) \in o(\log^* n)$. Independently, Czygrinow et al. showed by a related, but different approach that a constant approximation is impossible within $o(\log^* n)$ rounds [25].

On the other hand, unit disk graphs feature *bounded growth*, i.e., the maximal number of independent nodes in each $r$-hop neighborhood is polynomially bounded in $r$:

$$\forall v \in V, r \in \mathbb{N}: \max_{\text{IS } I \subseteq \mathcal{N}_v^{(r)}} \{|I|\} \leq f(r),$$

where $f$ is some polynomial. This property can be exploited in order to compute an MIS in $\mathcal{O}(\log^* n)$ deterministic rounds [98]. Note that graphs of bounded growth (and thus in particular unit disk graphs) also exhibit the weaker property of *bounded independence*, i.e., the neighborhood of each node contains a constantly bounded number of independent nodes only. This implies that an MIS (which is always a DS) is an $\mathcal{O}(1)$ MDS approximation, as the size of any IS is bounded by the size of an MDS times the respective constant. Therefore, our lower bound is matched asymptotically for the class of graphs of bounded growth. In contrast, the fastest known MIS algorithms on graphs of bounded independence are just the ones for general graphs, leaving a factor of $\mathcal{O}(\log n/\log^* n)$ between the strongest upper and lower bounds known so far. Note, however, that algorithms that are based on computing a MIS cannot be faster than $\Omega(\sqrt{\log n})$ rounds, as the lower bound by Kuhn et al. [54] applies to line graphs which feature bounded independence.

Complementary, in Chapter 12, we consider graphs of small arboricity. Such graphs can be very different from graphs with small independence. While in the latter case the number of edges may be large (in particular, in the complete graph there are no two independent nodes), graphs of constantly bounded arboricity have $\mathcal{O}(n)$ edges, but potentially large sets of independent neighbours (the star graph has arboricity one). We remark that it is not difficult to see that demanding both bounded growth and arboricity is equivalent to asking for bounded degree. In the family of graphs of maximum degree $\Delta \in \mathcal{O}(1)$ simply taking all nodes yields a trivial $\Delta + 1$ MDS approximation; we already mentioned that finding an MIS here takes $\Theta(\log^* n)$ rounds [40, 89].

We devise two algorithms for graphs of small arboricity $A$. The first employs a forest decomposition to obtain an $\mathcal{O}(A^2)$ MDS approximation within time $\mathcal{O}(\log n)$ w.h.p. This algorithm utilizes the fact that not too many nodes may be covered by their children in a given forest decomposition, implying that a good approximation ratio can be maintained if we cover all nodes that have one by a parent. Solving this problem approximatively can be reduced to computing an arbitrary MIS in some helper graph. Therefore, we get a randomized running time of $\mathcal{O}(\log n)$ w.h.p., whereas the approximation guarantee of $\mathcal{O}(A^2)$ is deterministic.

The second algorithm we propose is based on the property that subgraphs of graphs of bounded arboricity are sparse, i.e., if having $n'$ nodes, they contain at most $A(n'-1)$ edges. For this reason, we can build on the very simple greedy strategy of adding all nodes of locally large degree to the output set simultaneously, until eventually, after $\mathcal{O}(\log \Delta)$ rounds, all nodes are covered. This straightforward approach yields an $\mathcal{O}(A \log \Delta)$ approximation if one makes sure that the number of covered nodes in each repetition is at least the number of selected nodes. The latter can easily be done by requiring that uncovered nodes choose one of their eligible neighbors to enter the set instead of just electing all possible candidates into the set. Playing with the factor up to which joining nodes are required to have largest degree within their two-neighborhood, the algorithm can be modified to an $\mathcal{O}(\alpha A \log_\alpha \Delta)$ approximation within $\mathcal{O}(\log_\alpha \Delta)$ time, for any integer $\alpha \geq 2$. This second algorithm appeals by its simplicity; unlike the first, it is uniform, deterministic, and merely requires port numbers.

Recall that the best algorithm with polylogarithmically sized messages for general graphs has running time $\mathcal{O}(\log^2 \Delta)$ and approximation ratio $\mathcal{O}(\log \Delta)$ [56]. Thus, the algorithms given in Chapter 12 clearly improve on this result for graphs of bounded arboricity. However, although the lower bound from [54] fails to hold for such graphs, our algorithms' running times do not get below the thresholds of $\Omega(\sqrt{\log n})$ and $\Omega(\log \Delta)$, respectively. In contrast, we show in Chapter 13 that in planar graphs an $\mathcal{O}(1)$ approximation can be computed in $\mathcal{O}(1)$ rounds.[1] Our algorithm makes use of the facts that planar graphs and their minors are sparse, i.e., contain only $\mathcal{O}(n)$ edges, and that in planar graphs circles separate their interior (with respect to an embedding) from their outside.

A drawback of our algorithm for planar graphs is that it is rendered impractical because it relies on messages that in the worst case encode the whole graph. For the same reason, the technique by Czygrinow et al. [25] to obtain a $1 + \varepsilon$ approximation in $\mathcal{O}(\log^* n)$ rounds is also of theoretical significance only. If message size is required to be (poly)logarithmic in $n$, to the best of our knowledge the most efficient distributed algorithms are the ones from Chapter 12. More generally, the same is true for any graph family excluding fixed minors. Also here distributed $1 + \varepsilon$ approximations are known [23, 24], however of polylogarithmic running time with large exponent and again using large messages. Excluding particular minors is a rich source of graph families, apart from planar graphs including e.g. graphs of bounded genus or treewidth. Again to the best of our knowledge, currently no distributed algorithms tailored to these families of graphs exist. Moreover, as mentioned before, graphs of bounded (or non-constant, but slowly growing) arboricity extend beyond minor-free graph families. Therefore, the algorithms presented in Chapter 12 improve on the best known solutions for a wide range of inputs. See Table 9.2 for a comparison of distributed MDS approximations.

[1] This result was claimed previously, in [61] by us and independently in [25] by others. Sadly, both proofs turned out to be wrong.

Table 9.2: Upper and lower bounds on distributed MDS approximation in various graph families. For upper bounds, message size is stated to be "trivial" if the whole topology might be collected at individual nodes.

| graph family | running time | approximation ratio | deterministic | message size |
|---|---|---|---|---|
| general [56] | $\mathcal{O}(\log n)$ | $\mathcal{O}(\log \Delta)$ | no | trivial |
| general [56] | $\mathcal{O}(\log^2 \Delta)$ | $\mathcal{O}(\log \Delta)$ | no | $\mathcal{O}(\log \Delta)$ |
| general [54] | $\Omega\left(\min\left\{\sqrt{\log n}, \log \Delta\right\}\right)$ | polylog $n$ | no | arbitrary |
| bounded independence [80] | $\mathcal{O}(\log n)$ | $\mathcal{O}(1)$ | no | $\mathcal{O}(1)$ |
| bounded independence [90] | $2^{\mathcal{O}(\sqrt{\log n})}$ | $\mathcal{O}(1)$ | yes | trivial |
| bounded growth [98] | $\mathcal{O}(\log^* n)$ | $\mathcal{O}(1)$ | yes | $\mathcal{O}(\log n)$ |
| unit disk (Chapter 11) | $g(n) \in o(\log^* n)$ | $\notin o\left(\frac{\log^* n}{g(n)}\right)$ | yes | arbitrary |
| maximum degree $\Delta$ (trivial) | 0 | $\Delta + 1$ | yes | N/A |
| forests (trivial) | 1 | 3 | yes | $\mathcal{O}(1)$ |
| planar (Chapter 13) | 6 | 130 | yes | trivial |
| planar [25] | $\mathcal{O}(\log^* n)$ | $1 + \varepsilon$ | yes | trivial |
| excluded minor [23, 24] | polylog $n$ | $1 + \varepsilon$ | yes | trivial |
| arboricity $A$ (Chapter 12) | $\mathcal{O}(A^2)$ | $\mathcal{O}(\log n)$ | no | $\mathcal{O}(\log n)$ |
| arboricity $A$ (Chapter 12) | $\mathcal{O}(\log_\alpha \Delta)$ | $\mathcal{O}(\alpha A \log_\alpha \Delta)$ | yes | $\mathcal{O}(\log \log_\alpha \Delta)$ |

# Chapter 10

# Maximal Independent Sets on Trees

*"In fact, distributed MIS computation can be seen as a Drosophila of distributed computing as it prototypically models symmetry breaking [. . . ]" – Fabian Kuhn, The Price of Locality.*

In this chapter, we present a uniform randomized MIS algorithm that has a running time of $\mathcal{O}(\sqrt{\log n \log\log n})$ in forests w.h.p. Over each edge, $\mathcal{O}(\log n)$ bits are transmitted w.h.p. This material appears as [67].

## 10.1 Algorithm

For the sake of simplicity, we first describe a simplified variant of the algorithm, which $(i)$ is non-uniform, $(ii)$ makes use of uniformly random real numbers, and $(iii)$ whose pseudocode contains a generic term of $\Theta(R)$. In Theorem 10.10, we will show how to remove the first two assumptions, and it will turn out that for the uniform algorithm the precise choice of the constants in the term $\Theta(R)$ is of no concern (barring constant factors).

The algorithm seeks to increase the number of nodes in the independent set $I$ perpetually until it is maximal. Whenever a node enters $I$, its inclusive neighborhood is removed from the graph and the algorithm proceeds on the remaining subgraph of $G$. The algorithm consists of three main steps, each of which employs a different technique to add nodes to $I$. It takes a single parameter $R$, which ideally is small enough to guarantee a small running time of the first two loops of the algorithm, but large enough to guarantee that the residual nodes can be dealt with quickly by the final loop of the algorithm.

**Algorithm 10.1**: Fast MIS on Trees.

```
input  : $R \in \mathbb{N}$
output: MIS $I$
$I := \emptyset$
for $i \in \{1, \dots, \Theta(R)\}$ do // reduce degrees
    for $v \in V$ in parallel do
        $r_v :=$ u.i.r. number from $[0, 1] \subset \mathbb{R}$
        if $r_v > \max_{w \in \mathcal{N}_v}\{r_w\}$ then // join IS (no neighbor can)
            $I := I \cup \{v\}$
            delete $\mathcal{N}_v^+$ from $G$
        end
    end
end
for $i \in \{1, 2\}$ do // remove nodes of small degree
    $H :=$ subgraph of $G$ induced by nodes of degree $\delta_v \leq R$
    $(R + 1)$-color $H$
    for $i \in \{1, \dots, R + 1\}$ do
        for $v \in V$ with $\mathcal{C}(v) = i$ in parallel do // colors independent
            $I := I \cup \{v\}$
            delete $\mathcal{N}_v^+$ from $G$
        end
    end
end
while $V \neq \emptyset$ do // clean up
    for $v \in V$ in parallel do
        if $\delta_v \in \{0, 1\}$ then // remove isolated nodes and leaves
            if $\exists w \in \mathcal{N}_v$ with $\delta_w = 1$ then // adjacent leaves
                $r_v :=$ u.i.r. number from $[0, 1] \subset \mathbb{R}$
                if $r_v > r_w$ then
                    $I := I \cup \{v\}$
                    delete $\mathcal{N}_v^+$ from $G$
                end
            end
            else
                $I := I \cup \{v\}$
                delete $\mathcal{N}_v^+$ from $G$
            end
        end
    end
end
```

We proceed by describing the parts of the algorithm in detail. See Algorithm 10.1 for its pseudocode. In the first part, the following procedure is repeated $\Theta(R)$ times. Each active node draws a number from $[0,1] \subset \mathbb{R}$ u.i.r. and joins $I$ if its value is larger than the ones of all its neighbors.[1] Techniques similar to this one, which is due to Luby [73], have been known for a long time and reduce the number of edges in the graph exponentially w.h.p.; however, in our case we have the goal of reducing the maximum degree in the graph rapidly. Once the maximum degree is small, we cannot guarantee a quick decay w.h.p. anymore, therefore the employed strategy is changed after $\Theta(R)$ rounds.

Consequently, the second part of the algorithm aims at dealing with small-degree nodes according to a deterministic scheme. We will show that if $R$ is large enough, most nodes will not have more than $R$ neighbors of degree larger than $R$ in their neighborhood, even if not all nodes of degree larger than $R$ could be removed during the first loop. Thus, removing all nodes of degree at most $R$, we reduce for most nodes the degree to $R$. To this end, we first $(R+1)$-color the subgraph induced by all nodes of degree at most $R$ and then iterate through the colors, concurrently adding all nodes to $I$ that share a color. Executing this subroutine for a second time, all nodes that previously had at most $R$ neighbors of degree larger than $R$ will be eliminated.

Yet, a small fraction of the nodes may still remain active. To deal with these nodes, we repeat the step of removing all leaves and isolated nodes from the forest until all nodes have terminated.

As evident from the description of the algorithm, each iteration of the first or third loop can be implemented within a constant number of synchronous distributed rounds. The second loop requires $\mathcal{O}(R)$ time plus the number of rounds needed to color the respective subgraph; for this problem deterministic distributed algorithms taking $\mathcal{O}(R + \log^* n)$ time are known [8, 49]. In Theorem 10.9 we will show that for some $R \in \mathcal{O}(\sqrt{\log n \log\log n})$, the third loop of the algorithm will complete in $\mathcal{O}(R)$ rounds w.h.p. Thus, for an appropriate value of $R$, the algorithm computes an MIS within $\mathcal{O}(\sqrt{\log n \log\log n})$ rounds.

## 10.2 Analysis

For the purpose of our analysis, we will w.l.o.g. assume that the graph is a rooted tree. In order to execute the algorithm, the nodes do not need access to this information. The children of $v \in V$ are denoted by $C_v \subseteq \mathcal{N}_v$. The

[1]A random real number from $[0,1]$ can be interpreted as infinite string of bits of decreasing significance. As nodes merely need to know which one of two values is larger, it is sufficient to generate and compare random bits until the first difference occurs.

lion's share of the argumentation will focus on the first loop of Algorithm 10.1. We will call an iteration of this loop a *phase.* By $\delta_v(i)$ we denote the degree of node $v$ at the beginning of phase $i \in \{1, \dots, \Theta(R)\}$ in the subgraph of $G$ induced by the nodes that have not been deleted yet; similarly, $\mathcal{N}_v(i)$ and $C_v(i)$ are the sets of neighbors and children of $v$ that are still active at the beginning of phase $i$, respectively.

We start our analysis with the observation that in any phase, a high-degree node without many high-degree children is likely to be deleted in that phase, independently of the behavior of its parent.

**Lemma 10.1.** *Suppose that at the beginning of phase $i$ it holds for a node $v$ that half of its children have degree at most $\delta_v(i)/(16 \ln \delta_v(i))$. Then, independently of the random number of its parent, $v$ is deleted with probability at least $1 - 5/\delta_v(i)$ in that phase.*

*Proof.* Observe that the probability that $v$ survives phase $i$ is increasing in the degree $\delta_w(i)$ of any child $w \in C_v(i)$ of $v$. Thus, w.l.o.g., we may assume that all children of $v$ of degree at most $\delta := \delta_v(i)/(16 \ln \delta_v(i))$ have exactly that degree.

Consider such a child $w$. With probability $1/\delta$, its random value $r_w(i)$ is larger than all its children's. Denote by $X$ the random variable counting the number of children $w \in C_v(i)$ of degree $\delta$ satisfying that

$$\forall u \in C_w(i): \; r_w(i) > r_u(i). \tag{10.1}$$

It holds that

$$\mathbb{E}[X] = \sum_{\substack{w \in C_v(i) \\ \delta_w(i) = \delta}} \frac{1}{\delta} \geq \frac{\delta_v}{2\delta} \geq 8 \ln \delta_v(i).$$

Since the random choices are independent, applying Corollary 2.13 yields that

$$P\left[X < \frac{\mathbb{E}[X]}{2}\right] < e^{-\mathbb{E}[X]/8} \leq \frac{1}{\delta_v(i)}.$$

Node $v$ is removed unless the event $\mathcal{E}$ that $r_v(i) > r_w(i)$ for all the children $w \in C_v(i)$ of degree $\delta$ satisfying (10.1) occurs. If $\mathcal{E}$ happens, this implies that $r_v(i)$ is also greater than all random values of children of such $w$, i.e., $r_v(i)$ is greater than $\delta\, \mathbb{E}[X]/2 \geq \delta_v(i)/4$ other independent random values. Since the event that $X \geq \mathbb{E}[X]/2$ depends only on the order of the involved random values, we infer that $P[\mathcal{E} \,|\, X \geq \mathbb{E}[X]/2] < 4/\delta_v(i)$. We conclude that $v$ is deleted with probability at least

$$P\left[X \geq \frac{\mathbb{E}[X]}{2}\right] P\left[\bar{\mathcal{E}} \,\middle|\, X \geq \frac{\mathbb{E}[X]}{2}\right] > \left(1 - \frac{1}{\delta_v(i)}\right)\left(1 - \frac{4}{\delta_v(i)}\right) > 1 - \frac{5}{\delta_v(i)}$$

as claimed. Since we reasoned about whether children of $v$ join the independent set only, this bound is independent of the behavior of $v$'s parent. □

Applied inductively, this result implies that in order to maintain a high degree for a considerable number of phases, a node must be the root of a large subtree. This concept is formalized by the following definition.

**Definition 10.2** (Delay Trees)**.** *A* delay tree *of depth $d \in \mathbb{N}_0$ rooted at node $v \in V$ is defined recursively as follows. For $d = 0$, the tree consists of $v$ only. For $d > 0$, node $v$ satisfies at least one of the following criteria:*

*(i) At least $\delta_v(d)/4$ children $w \in C_v$ are roots of delay trees of depth $d-1$ and have $\delta_w(d-1) \geq \delta_v(d)/(16 \ln \delta_v(d))$.*

*(ii) Node $v$ is the root of a delay tree of depth $d-1$ and $\delta_v(d-1) \geq \delta_v(d)^2/(324 \ln \delta_v(d))$.*

In order to bound the number of phases for which a node has a high chance of retaining a large degree, we bound the depth of delay trees rooted at high-degree nodes.

**Lemma 10.3.** *Assume that $R \geq 2\sqrt{\ln n \ln \ln n}$ and $\delta_v(d) \geq e^R$. Then for a delay tree of depth $d$ rooted at $v$ it holds that $d \in \mathcal{O}(\sqrt{\log n / \log \log n})$.*

*Proof.* Assume w.l.o.g. that $d > 1$. Denote by $s_i(\delta)$, where $i \in \{0, \ldots, d-1\}$ and $\delta \in \mathbb{N}$, the minimal number of leaves in a delay tree of depth $i$ rooted at some node $w$ satisfying $\delta_w(i+1) = \delta$.

We claim that for any $\delta$ and $i \leq \ln \delta / (2 \ln(324 \ln \delta))$, it holds that

$$s_i(\delta) \geq \prod_{j=1}^{i} \frac{\delta}{(324 \ln \delta)^{j-1}},$$

which we show by induction. This is trivially true for $i = 1$, hence we need to perform the induction step only. For any $i \in \{2, \ldots, \lfloor \ln \delta / (2 \ln(324 \ln \delta)) \rfloor\}$, the assumption that the claim is true for $i-1$ and the recursive definition of delay trees yield that

$$\begin{aligned}
s_i(\delta) &\geq \min\left\{\frac{\delta}{4}s_{i-1}\left(\frac{\delta}{16\ln\delta}\right), s_{i-1}\left(\frac{\delta^2}{324\ln\delta}\right)\right\}\\
&\geq \min\left\{\frac{\delta}{4}\prod_{j=1}^{i-1}\frac{\delta/(16\ln\delta)}{(16\ln(\delta/(16\ln\delta)))^{j-1}}, \prod_{j=1}^{i-1}\frac{\delta^2/(324\ln\delta))}{(324\ln(\delta/(324\ln\delta)))^{j-1}}\right\}\\
&> \min\left\{\frac{\delta}{4}\prod_{j=1}^{i-1}\frac{\delta}{(16\ln\delta)^j}, \prod_{j=1}^{i-1}\frac{\delta^2}{(324\ln\delta)^j}\right\}\\
&> \delta\prod_{j=1}^{i-1}\frac{\delta}{(324\ln\delta)^j}\\
&= \prod_{j=1}^{i}\frac{\delta}{(324\ln\delta)^{j-1}}.
\end{aligned}$$

Thus the induction step succeeds, showing the claim.

Now assume that $v$ is the root of a delay tree of depth $d-1$. As $\delta_v(d) \geq e^R$, we may insert any $i \in \{1,\ldots,\min\{d-1,\lfloor R/(2\ln 324R)\rfloor\}\}$ into the previous claim. Supposing for contradiction that $d-1 \geq \lfloor R/(2\ln 324R)\rfloor$, it follows that the graph contains at least

$$\prod_{j=1}^{\lfloor R/(2\ln 324R)\rfloor}\frac{e^R}{(324R)^{j-1}} > \prod_{j=1}^{\lfloor R/(2\ln 324R)\rfloor} e^{R/2} \in e^{R^2/((4+o(1))\ln R)} \subseteq n^{2-o(1)}$$

nodes. On the other hand, if $d-1 < \lfloor R/(2\ln 324R)\rfloor$, we get that the graph contains at least $e^{(d-1)R/2}$ nodes, implying that $d \in \mathcal{O}(\sqrt{\ln n/\ln\ln n})$ as claimed. □

With this statement at hand, we infer that for $R \in \Theta(\sqrt{\log n \log\log n})$, it is unlikely that a node has degree $e^R$ or larger for $R$ phases.

**Lemma 10.4.** *Suppose that $R \geq 2\sqrt{\ln n \ln\ln n}$. Then, for any node $v \in V$ and some number $r \in \mathcal{O}(\sqrt{\log n/\log\log n})$, we have that $\delta_v(r+1) < e^R$ with probability at least $1-6e^{-R}$. This statement holds independently of the behavior of v's parent.*

*Proof.* Assume that $\delta_v(r) \geq e^R$. According to Lemma 10.1, node $v$ is removed with probability at least $1-5/\delta_v(r)$ in phase $r$ unless half of its children have at least degree $\delta_v(r)/(16\ln\delta_v(r))$.

Suppose the latter is the case and that $w$ is such a child. Applying Lemma 10.1, we see that in phase $r-1$, when we have $\delta_w(r-1) \geq \delta_w(r) \geq \delta_v(r)/(16\ln\delta_v(r))$, $w$ is removed with probability $1-5/\delta_w(r-1)$ if it does not have $\delta_w(r-1)/2$ children of degree at least $\delta_w(r-1)/(16 \ln \delta_w(r-1))$. Thus, the expected number of such nodes $w$ that do not themselves have many high-degree children in phase $r-1$ but survive until phase $r$ is bounded by

$$\frac{5\delta_v(r-1)}{\delta_w(r-1)} \leq \frac{80\delta_v(r-1)\ln \delta_v(r)}{\delta_v(r)}.$$

Since Lemma 10.1 states that the probability bound for a node $w \in C_v(r-1)$ to be removed is independent of $v$'s actions, we can apply Corollary 2.13 in order to see that

$$\frac{(80+1/2)\delta_v(r-1)\ln \delta_v(r)}{\delta_v(r)} + \mathcal{O}(\log n)$$

of these nodes remain active at the beginning of phase $r$ w.h.p. If this number is not smaller than $\delta_v(r)/4 \in \omega(\log n)$, it holds that $\delta_v(r-1) \in \delta_v(r)^2/(4(80+1/2+o(1))\ln \delta_v(r))$. Otherwise, at least $\delta_v(r)/2 - \delta_v(r)/4 = \delta_v(r)/4$ children $w \in C_v(r-1)$ have degree $\delta_w(r-1) \geq \delta_v(r)/16 \ln \delta_v(r)$. In both cases, $v$ meets one of the conditions in the recursive definition of a delay tree. Repeating this reasoning inductively for all $r \in \mathcal{O}(\sqrt{\log n/\log\log n})$ rounds (where we may choose the constants in the $\mathcal{O}$-term to be arbitrarily large), we construct a delay tree of depth at least $r$ w.h.p.

However, Lemma 10.3 states that $r$ must be in $\mathcal{O}(\sqrt{\log n/\log\log n})$ provided that $\delta_v(r) \geq e^R$. Therefore, for an appropriate choice of constants, we conclude that the event $\mathcal{E}$ that both half of the nodes in $C_v(r)$ have degree at least $\delta_v(r)/16 \ln \delta_v(r)$ *and* $\delta_v(r) \geq e^R$ w.h.p. does not occur. If $\mathcal{E}$ does not happen, but $\delta_v(r) \geq e^R$, Lemma 10.1 gives that $v$ is deleted in phase $r$ with probability at least $1 - 5e^{-R}$.

Thus, the total probability that $v$ is removed or has sufficiently small degree at the beginning of phase $r+1$ is bounded by

$$\begin{aligned} & P\left[\bar{\mathcal{E}}\right] \cdot P\left[v \text{ deleted in phase } r \,\middle|\, \bar{\mathcal{E}} \text{ and } \delta_v(r) \geq e^R\right] \\ > \; & \left(1 - \frac{1}{n^c}\right)\left(1 - \frac{5}{e^R}\right) \\ > \; & 1 - 6e^{-R}, \end{aligned}$$

where we used that $R < \ln n$ because $\delta_v \leq n-1$. Since all statements used hold independently of $v$'s parent's actions during the course of the algorithm, this concludes the proof. □

For convenience, we rephrase the previous lemma in a slightly different way.

**Corollary 10.5.** *Provided that $R \geq 2\sqrt{\log n \log\log n}$, for any node $v \in V$ it holds with probability $1 - e^{-\omega(R)}$ that $\delta_v(R) < e^R$. This bound holds independently of the actions of a constant number of $v$'s neighbors.*

*Proof.* Observe that $R \in \omega(r)$, where $r$ and $R$ are defined as in Lemma 10.4. The lemma states that after $r$ rounds, $v$ retains $\delta_v(r+1) \geq e^R$ with probability at most $6e^{-R}$. As the algorithm behaves identically on the remaining subgraph, we can apply the lemma repeatedly, giving that $\delta_v(R) < e^R$ with probability $1 - e^{-\omega(R)}$. Ignoring a constant number of $v$'s neighbors does not change the asymptotic bounds. □

Since we strive for a sublogarithmic value of $R$, the above probability bound does not ensure that all nodes will have degree smaller than $e^R$ after $R$ phases w.h.p. However, on paths of length at least $\sqrt{\ln n}$, at least one of the nodes will satisfy this criterion w.h.p. Moreover, nodes of degree smaller than $e^R$ will have left a few high-degree neighbors only, which do not interfere with our forthcoming reasoning.

**Lemma 10.6.** *Assume that $R \geq 2\sqrt{\log n \log\log n}$. Given some path $P = (v_0, \ldots, v_k)$, define for $i \in \{0, \ldots, k\}$ that $G_i$ is the connected component of $G$ containing $v_i$ after removing the edges from $P$. If $\delta_{v_i}(R) < e^R$, denote by $\bar{G}_i$ the connected (sub)component of $G_i$ consisting of nodes $w$ of degree $\delta_w(R) < e^R$ that contains $v_i$. Then, with probability $1 - e^{-\omega(\sqrt{\ln n})}$, we have that*

*(i)* $\delta_{v_i}(R) < e^R$ *and*

*(ii) nodes in $\bar{G}_i$ have at most $\sqrt{\ln n}$ neighbors $w$ of degree $\delta_w(R) \geq e^R$.*

*This probability bound holds independently of anything that happens outside of $G_i$.*

*Proof.* Corollary 10.5 directly yields Statement $(i)$. For the second statement, let $u$ be any node of degree $\delta_u(R) < e^R$. According to Corollary 10.5, all nodes $w \in C_u(R)$ have $\delta_w(R) < e^R$ with independently bounded probability $1 - e^{-\omega(R)}$. In other words, the random variable counting the number of such nodes having degree $\delta_w(R) \geq e^R$ is stochastically dominated from above by the sum of $\delta_u(R)$ independent Bernoulli variables attaining the value 1 with probability $e^{-\omega(R)} \subset e^{-\omega(\sqrt{\ln n})}$. Applying Corollary 2.13, we conclude that w.h.p. no more than $\sqrt{\ln n}$ of $u$'s neighbors have a degree that is too large. Due to the union bound, thus both statements are true with probability $1 - e^{-\omega(\sqrt{\ln n})}$. □

Having dealt with nodes of degree $e^R$ and larger, we need to show that we can get rid of the remaining nodes sufficiently fast. As a first step, we show a result along the lines of Lemma 10.1, trading in a weaker probability bound for a stronger bound on the degrees of a node's children.

**Lemma 10.7.** *Given a constant $\beta \in \mathbb{R}^+$, assume that in phase $i \in \mathbb{N}$ for a node $v \in V$ we have that at least $e^{-\beta}\delta_v(i)$ of its children have degree at most $e^{\beta}\delta_v(i)$. Then $v$ is deleted with at least constant probability in that phase, regardless of the random value of its parent.*

*Proof.* As in Lemma 10.1, we may assume w.l.o.g. that all children of $v$ with degree at most $\delta := e^{\beta}\delta_v(i)$ have exactly that degree. For the random variable $X$ counting the number of nodes $w \in C_v(i)$ of degree $\delta$ that satisfy Condition 10.1 we get that

$$\mathbb{E}[X] = \sum_{\substack{w \in \mathcal{C}_v(i) \\ \delta_w(i) = \delta}} \frac{1}{\delta} \geq e^{-2\beta}.$$

Since the random choices are independent, applying Chernoff's bound yields that

$$P[X = 0] \leq e^{-\mathbb{E}[X]/2} < e^{-e^{-2\beta}/2} =: \gamma.$$

Provided that $X > 0$, there is at least one child $w \in C_v$ of $v$ that joins the set in phase $i$ unless $r_v(i) > r_w(i)$. Since $r_w(i)$ is already larger than all of its neighbors' random values (except maybe $v$), the respective conditional probability $P[r_v(i) > r_w(i) \,|\, w \text{ satisfies Inequality (10.1)}]$ certainly does not exceed $1/2$, i.e.,

$$P[X > 0] \cdot P[v \text{ is deleted in phase } i \,|\, X > 0] \geq \frac{1-\gamma}{2}.$$

Since we reasoned about whether children of $v$ join the independent set exclusively, this bound is independent of the behavior of $v$'s parent. □

We cannot guarantee that the maximum degree in the subgraph formed by the active nodes drops quickly. However, we can show that for all but a negligible fraction of the nodes this is the case.

**Lemma 10.8.** *Denote by $H = (V_H, E_H)$ a subgraph of $G$ still present in phase $R$ in which all nodes have degree smaller than $e^R$ and for any node no more than $\mathcal{O}(\sqrt{\log n})$ neighbors outside $H$ are still active in phase $R$. If $R \geq R(n) \in \mathcal{O}(\sqrt{\log n \log\log n})$, it holds that all nodes from $H$ are deleted after the second for-loop of the algorithm w.h.p.*

*Proof.* For the sake of simplicity, we consider the special case that no edges to nodes outside $H$ exist first. We claim that for a constant $\alpha \in \mathbb{N}$ and all $i, j \in \mathbb{N}_0$ such that $i > j$ and $e^{R-j} \geq 8ec \ln n$, it holds that

$$\max_{v \in V} \left\{ \left| \left\{ w \in C_v(R + \alpha i) \,|\, \delta_w(R + \alpha i) > e^{R-j} \right\} \right| \right\} \leq \max \left\{ e^{R-2i+j}, 8c \ln n \right\}$$

w.h.p. For $i = 1$ we have $j = 0$, i.e., the statement holds by definition because degrees in $H$ are bounded by $e^R$. Assume the claim is established for some value of $i \geq 1$.

Consider a node $w \in H$ of degree $\delta_w(R + \alpha i) > e^{R-j} \geq 8ec \ln n$ for some $j \leq i$. By induction hypothesis, the number of children of $w$ having degree larger than $e^{R-(j-1)}$ in phase $R + \alpha i$ (and thus also subsequent phases) is bounded by $\max\{e^{R-(2i-(j-1))}, 8c \ln n\} \leq e^{R-(j+1)}$ w.h.p., i.e., at least a fraction of $1 - 1/e$ of $w$'s neighbors has degree at most factor $e$ larger than $\delta_w(R + \alpha i)$ in phase $R + \alpha i$. According to Lemma 10.7, this implies that $w$ is removed with constant probability in phase $R + \alpha i$. Moreover, as long as $w$ has such a high degree, there is at least a constant probability that $w$ is removed in each subsequent phase. This constant probability bound holds independently from previous phases (conditional to the event that $w$ retains degree larger than $e^{R-j}$). Furthermore, due to the lemma, it applies to all children $w$ of a node $v \in H$ independently. Hence, applying Corollary 2.13, we get that in all phases $k \in \{\alpha i, \alpha i + 1, \ldots, \alpha(i + 1) - 1\}$, the number $|\{w \in C_v(k) \,|\, \delta_w(k) > e^{R-j}\}|$ is reduced by a constant factor w.h.p. (unless this number is already smaller than $8c \ln n$). Consequently, if the constant $\alpha$ is sufficiently large, the induction step succeeds, completing the induction.

Recapitulating, after in total $\mathcal{O}(R)$ phases, no node in $H$ will have more than $\mathcal{O}(\log n)$ neighbors of degree larger than $\mathcal{O}(\log n)$. The previous argument can be extended to reduce degrees even further. The difficulty arising is that once the expected number of high-degree nodes removed from the respective neighborhoods becomes smaller than $\Omega(\log n)$, Chernoff's bound does no longer guarantee that a constant fraction of high-degree neighbors is deleted in each phase. However, as used before, for critical nodes the applied probability bounds hold in each phase independently of previous rounds. Thus, instead of choosing $\alpha$ as a constant, we simply increase $\alpha$ with $i$.

Formally, if $j_0$ is the last index such that $e^{R-j} \geq 8ec \ln n$, we define

$$\alpha(i) := \begin{cases} \alpha & \text{if } i \leq j_0 \\ \lceil \alpha e^{i-j_0} \rceil & \text{otherwise.} \end{cases}$$

This way, the factor $e$ loss in size of expected values (weakening the outcome of Chernoff's bound) is compensated for by increasing the number of considered phases by factor $e$ (which due to independence appears in the exponent of the bound) in each step. Hence, within

$$\sum_{i=\lceil \ln \sqrt{\log n}\,\rceil}^{R} \alpha(i) \in \mathcal{O}\left(R + \sum_{i=\lceil \ln \sqrt{\log n}\rceil}^{\lceil \ln(8ec\ln n)\rceil} \frac{\ln n}{e^i}\right) = \mathcal{O}\left(R + \frac{\ln n}{\sqrt{\log n}}\right) = \mathcal{O}(R)$$

phases no node in $H$ will have left more than $\mathcal{O}(\sqrt{\log n})$ neighbors of degree larger than $\mathcal{O}(\sqrt{\log n})$ w.h.p. Assuming that constants are chosen appropriately, this is the case after the first for-loop of the algorithm.

Recall that in the second loop the algorithm removes all nodes of degree at most $R$ in each iteration. Thus, degrees in $H$ are reduced to $\mathcal{O}(\sqrt{\log n})$ in the first iteration of the loop, and subsequently all remaining nodes from $H$ will be removed in the second iteration. Hence, indeed all nodes from $H$ are deleted at the end of the second for-loop w.h.p. as claimed.

Finally, recall that no node has more than $\mathcal{O}(\sqrt{\log n})$ edges to nodes outside $H$. Choosing constants properly, these edges contribute only a negligible fraction to the nodes' degrees even once these degrees reach $\mathcal{O}(\sqrt{\log n})$. Thus, the asymptotic statement obtained by the above reasoning holds true also if we consider a subgraph $H$ where nodes have $\mathcal{O}(\sqrt{\log n})$ edges leaving the subgraph. Thus the proof concludes. □

We can now prove our bound on the running time of Algorithm 10.1.

**Theorem 10.9.** *Assume that $G$ is a forest and the coloring steps of Algorithm 10.1 are performed by a subroutine running for $\mathcal{O}(R + \log^* n)$ rounds. Then the algorithm eventually terminates and outputs a maximal independent set. Furthermore, if $R \geq R(n) \in \mathcal{O}(\sqrt{\log n \log\log n})$, Algorithm 10.1 terminates within $\mathcal{O}(R)$ rounds w.h.p.*

*Proof.* Correctness is obvious because $(i)$ adjacent nodes can never join $I$ concurrently, $(ii)$ the neighborhoods of nodes that enter $I$ are deleted immediately, $(iii)$ no nodes from $V \setminus (\cup_{v\in I}\mathcal{N}_v^+)$ get deleted, and $(iv)$ the algorithm does not terminate until $V = \emptyset$. The algorithm will terminate eventually, as in each iteration of the third loop all leaves and isolated nodes are deleted and any forest contains either of the two.

Regarding the running time, assume that $R \in \mathcal{O}(\sqrt{\log n \log\log n})$ is sufficiently large, root the tree at an arbitrary node $v_0 \in V$, and consider any path $P = (v_0, \dots, v_k)$ of length $k \geq \sqrt{\ln n}$. Denote by $G_i$, $i \in \{0, \dots, k\}$, the connected component of $G$ containing $v_i$ after removing the edges of $P$ and—provided that $\delta_{v_i}(R) < e^R$—by $\bar{G}_i$ the connected (sub)component of $G_i$ that contains $v_i$ and consists of nodes $w$ of degree $\delta_w(R) < e^R$ (as in Lemma 10.6). Then, we have by Lemma 10.6 for each $i$ with a probability that is independently lower bounded by $1 - e^{-\omega(\sqrt{\ln n})}$ that

$(i)$ $\delta_{v_i}(R) < e^R$ and

$(ii)$ nodes in $\bar{G}_i$ have at most $\sqrt{\ln n}$ neighbors $w$ of degree $\delta_w(R) \geq e^R$.

In other words, for each $i$, with probability independently bounded by $1 - e^{-\omega(\sqrt{\ln n})}$, $\bar{G}_i$ exists and satisfies the prerequisites of Lemma 10.8, implying that all nodes in $\bar{G}_i$ are deleted by the end of the second for-loop w.h.p. We conclude that independently of all $v_j \neq v_i$, node $v_i$ is *not* deleted until the end of the second loop with probability $e^{-\omega(\sqrt{\ln n})}$. Thus, the probability that no node on $P$ is deleted is at most

$$\left(e^{-\omega(\sqrt{\ln n})}\right)^k \subseteq e^{-\omega(\ln n)} = n^{-\omega(1)}.$$

Hence, when the second loop is completed, w.h.p. no path of length $k \geq \sqrt{\ln n}$ starting at $v_0$ exists in the remaining graph. Since $v_0$ was arbitrary, this immediately implies that w.h.p. after the second loop no paths of length $k \geq \sqrt{\ln n}$ exist anymore, i.e., the components of the remaining graph have diameter at most $\sqrt{\ln n}$. Consequently, it will take at most $\sqrt{\ln n}$ iterations of the third loop until all nodes have been deleted.

Summing up the running times for executing the three loops of the algorithm, we get that it terminates within $\mathcal{O}(R + (R + \log^* n) + \sqrt{\ln n}) = \mathcal{O}(R)$ rounds w.h.p. □

We complete our analysis by deducing a uniform algorithm featuring the claimed bounds on time and bit complexity.

**Theorem 10.10.** *A uniform MIS algorithm exists that terminates on general graphs within $\mathcal{O}(\log n)$ rounds and on forests within $\mathcal{O}(\sqrt{\log n \log\log n})$ rounds, both w.h.p. It can be implemented such that $\mathcal{O}(\log n)$ bits are sent over each link w.h.p.*

*Proof.* Instead of running Algorithm 10.1 directly, we wrap it into an outer loop trying to guess a good value for $R$ (i.e., $R(n) \leq R \in \mathcal{O}(R(n))$, where $R(n)$ as in Theorem 10.9). Furthermore, we restrict the number of iterations of the third loop to $R$, i.e., the algorithm will terminate after $\mathcal{O}(R)$ steps, however, potentially without producing an MIS.[2] Starting e.g. from two, with each call $R$ is doubled. Once $R$ reaches $R(n)$, according to Theorem 10.9 the algorithm outputs an MIS w.h.p. provided that $G$ is a forest. Otherwise, $R$ continues to grow until it becomes logarithmic in $n$. At this point, the analysis of Luby's algorithm by Métivier et al. [80] applies to the first loop of our algorithm, showing that it terminates and return an MIS w.h.p. Hence,

[2]There is no need to start all over again; one can build on the IS of previous iterations, although this does not change the asymptotic bounds.

as the running time of each iteration of the outer loop is (essentially) linear in $R$ and $R$ grows exponentially, the overall running time of the algorithm is $\mathcal{O}(\sqrt{\log n \log\log n})$ on forests and $\mathcal{O}(\log n)$ on arbitrary graphs w.h.p.

Regarding the bit complexity, consider the first and third loop of Algorithm 10.1 first. In each iteration, a constant number of bits for state updates (entering MIS, being deleted without joining MIS, etc.) needs to be communicated as well as a random number that has to be compared to each neighbor's random number. However, in most cases exchanging a small number of leading bits is sufficient to break symmetry. Overall, as shown by Métivier et al. [80], this can be accomplished with a bit complexity of $\mathcal{O}(\log n)$ w.h.p. Essentially, for every round their algorithm generates a random value and transfers the necessary number of leading bits to compare these numbers to each neighbor only. By Corollary 2.13, comparing $\mathcal{O}(\log n)$ random numbers between neighbors thus requires $\mathcal{O}(\log n)$ exchanged bits w.h.p., as in expectation each comparison requires to examine a constant number of bits. Thus, if nodes do not wait for a phase to complete, but rather continue to exchange random bits for future comparisons in a stream-like fashion, the bit complexity becomes $\mathcal{O}(\log n)$.

However, in order to avoid increasing the (sublogarithmic) time complexity of the algorithm on forests, more caution is required. Observe that in each iteration of the outer loop, we know that $\Theta(R)$ many random values need to be compared to execute the respective call of Algorithm 10.1 correctly. Thus, nodes may exchange the leading bits of these $\Theta(R)$ many random values concurrently, without risking to increase the asymptotic bit complexity. Afterwards, for the fraction of the values for which the comparison remains unknown, nodes send the second and the third bit to their neighbors simultaneously. In subsequent rounds, we double the number of sent bits per number repeatedly. Note that for each single value, this way the number of sent bits is at most doubled, thus the probabilistic upper bound on the total number of transmitted bits increases at most by a factor of two. Moreover, after $\log\log n$ rounds, $\log n$ bits of each single value will be compared in a single round, thus at the latest after $\log\log n + \mathcal{O}(1)$ rounds all comparisons are completed w.h.p. Employing this scheme, the total time complexity of all executions of the first and third loop of Algorithm 10.1 is (in a forest) bounded by

$$\begin{aligned}\mathcal{O}\left(\sum_{i=1}^{\lceil \log R(n)\rceil}\left(2^i + \log\log n\right)\right) &\subseteq \mathcal{O}(R(n) + \log R(n)\log\log n)\\ &= \mathcal{O}\left(\sqrt{\log n \log\log n}\right)\end{aligned}$$

w.h.p.

It remains to show that the second loop of Algorithm 10.1 does not require the exchange of too many bits. The number of transmitted bits to execute this loop is determined by the number of bits sent by the employed coloring algorithm. Barenboim and Elkin [8] and Kuhn [49] independently provided deterministic coloring algorithms with running time $\mathcal{O}(R + \log^* n)$. These algorithms start from an initial coloring with a number of colors that is polynomial in $n$ (typically one assumes identifiers of size $\mathcal{O}(\log n)$), which can be obtained by choosing random colors from the range $\{1, \ldots, n^{\mathcal{O}(1)}\}$ w.h.p. Exchanging these colors (which also permits to verify that the random choices indeed resulted in a proper coloring) thus costs $\mathcal{O}(\log n)$ bits.[3] However, as the maximum degree of the considered subgraphs is $R + 1$, which is in $\mathcal{O}(\log n)$ w.h.p., subsequent rounds of the algorithms deal with colors that are of (poly)logarithmic size in $n$. As exchanging coloring information is the dominant term contributing to message size in both algorithms, the overall bit complexity of all executions of the second loop of Algorithm 10.1 can be kept as low as $\mathcal{O}(\log n + R(n) \log\log n) = \mathcal{O}(\log n)$. $\square$

[3]To derive a uniform solution, one again falls back to doubling the size of the bit string of the chosen color until the coloring is locally feasible.

Chapter 11

# A Lower Bound on Minimum Dominating Set Approximations in Unit Disk Graphs

*"Whoever solves one of these problems gets a bag of Swiss chocolate." – An incentive Roger offered for devising the lower bound presented in this chapter.*

In this chapter, we will show that in unit disk graphs, no deterministic distributed algorithm can compute an $f(n)$ MDS approximation in $g(n)$ rounds for any $f$, $g$ with $f(n)g(n) \in o(\log^* n)$. This bound holds even if message size is unbounded, nodes have unique identifiers, and the nodes know $n$. This chapter is based on [65].

## 11.1 Definitions and Preliminary Statements

The lower bound proof will reason about the following highly symmetric graphs.

**Definition 11.1** ($R_n^k$)**.** *For $k, n \in \mathbb{N}$, we define the* $k$-ring with $n$ nodes $R_n^k := (V_n, E_n^k)$ *by*

$$\begin{aligned} V_n &:= \{1, \ldots, n\} \\ E_n &:= \left\{ \{i,j\} \in \binom{V_n}{2} \;\middle|\; |(i-j) \bmod n| \leq k \right\}. \end{aligned}$$

*See Figure 11.1 for an illustration. By $R_n := R_n^1$ we denote the "simple" ring. Moreover, we will take numbers modulo $n$ when designating nodes on the ring, e.g. we identify $3n + 5 \equiv 5 \in V_n$.*

Obviously, for any $k$ and $n$ this graph is a UDG (see Definition 9.7).

**Lemma 11.2.** *$R_n^k$ can be realized as a UDG.*

*Proof.* For $n > 2k + 1$, place all nodes equidistantly on a circle of radius $1/(2\sin(k\pi/n))$. Otherwise, use any circle of radius at most $1/2$. □

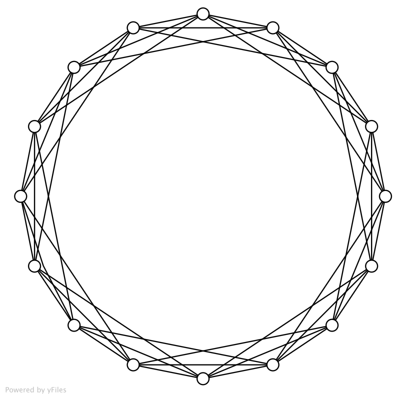

Figure 11.1: $R_{16}^3$. Realized as UDG $k$ is controlled by the scaling.

Our bound will be inferred from a classic result by Linial [70], which was later generalized to randomized algorithms by Naor [89].

**Theorem 11.3.** *There is no deterministic distributed algorithm 3-coloring the ring $R_n$ in fewer than $\frac{1}{2}(\log^* n - 1)$ communication rounds, even if the nodes know $n$.*

*Proof.* See e.g. [91]. □

We will use the following notion, which captures the amount of symmetry breaking information in the output of an algorithm.

**Definition 11.4** ($\sigma(n)$-Alternating Algorithms)**.** *Suppose $\mathcal{A}$ is an algorithm operating on $R_n$ which assigns each node $i \in V_n$ a binary output $b(i) \in \{0,1\}$. We call $\mathcal{A}$ $\sigma(n)$-*alternating*, if the length $\ell$ of any monochromatic sequence $b(i) = b(i+1) = \ldots = b(i+\ell)$ in the output of $\mathcal{A}$ is smaller than $\sigma(n)$.*

If a $\sigma(n)$-alternating algorithm is given, one can easily obtain a 3-coloring of the ring $R_n$ in $\mathcal{O}(\sigma(n))$ time.

**Lemma 11.5.** *Given a $\sigma(n)$-alternating algorithm $\mathcal{A}$ running in $\mathcal{O}(\sigma(n))$ rounds, a 3-coloring of the ring can be computed in $\mathcal{O}(\sigma(n))$ rounds.*

*Proof Sketch.* Essentially, nodes simply need to find the closest switch from 0 to 1 (or vice versa) in the output bits. From there, nodes are colored alternatingly, while the third color is used to resolve conflicts where the alternating sequences meet. One has to respect the fact that nodes do not agree on "left" and "right", though. See [65] for details. □

## 11.2 Proof of the Lower Bound

To establish our lower bound, we construct a $\sigma(n)$-alternating algorithm using an MDS approximation algorithm.

**Lemma 11.6.** *Assume a deterministic $f(n)$ approximation algorithm $\mathcal{A}$ for the MDS problem on UDG's running in at most $g(n) \geq 1$ rounds is given, where $f(n)g(n) \in o(\log^* n)$. Then an $o(\log^* n)$-alternating algorithm $\mathcal{A}'$ requiring $o(\log^* n)$ communication rounds exists.*

*Proof.* Assume w.l.o.g. that identifiers on $R_n^k$ are from $\{1,\ldots,n\}$. Consider $R_n^k$ and label each node $i \in V_n$ with its input $l(i)$, i.e., its identifier. Set $\sigma_k(n) := \max\{f(n),k\}g(n)$ and define

$$\begin{aligned}\mathcal{L}_n^k \quad := \quad & \Big\{ \big(l_1,\ldots,l_{\sigma_k(n)+2kg(n)}\big) \in \Big\{\{1,\ldots,n\}^{\sigma_k(n)+2kg(n)} \,|\, l_i \neq l_j \forall i \neq j\Big\} \\ & \mid \big(l(1) = l_1 \wedge \ldots \wedge l(\sigma_k(n)+2kg(n)) = l_{\sigma_k(n)+2kg(n)}\big) \\ & \Rightarrow \Big(b(kg(n)+1) = \ldots = b(\sigma_k(n)+kg(n)) = 1 \text{ on } R_n^k\Big)\Big\},\end{aligned}$$

i.e., the set of sequences of identifiers such that $\sigma_k(n)$ consecutive nodes will take the decision $b(v) = 1$ when $\mathcal{A}$ is executed on $R_n^k$, where the choices of the leading and trailing $kg(n)$ nodes may also depend on labels not in the considered sequence. As the decision of any node $i \in V_n$ depends on identifiers of nodes in $\mathcal{N}_i^{(kg(n))}$ only because it cannot learn from information further away in $g(n)$ rounds of communication on $R_n^k$, $\mathcal{L}_n^k$ is well-defined.

We distinguish two cases. The first case assumes that values $k_0, n_0 \in \mathbb{N}$ exist, such that for $n \geq n_0$ at most $n/2$ identifiers can simultaneously participate in disjoint sequences from $\mathcal{L}_n^{k_0}$ in a valid labeling of $R_n^{k_0}$ (where no identifier is used twice). Thus, for $n' := \max\{n_0, 2n\}$, an injective mapping $\lambda_n : \{1, \dots, n\} \to \{1, \dots, n'\}$ exists such that no element of $\mathcal{L}_{n'}^{k_0}$ is completely contained in the image of $\lambda_n$. Therefore, we can define $\mathcal{A}'$ to simulate a run of $\mathcal{A}$ on $R_{n'}^{k_0}$ where node $i \in \{1, \dots, n\}$ is labeled by $\lambda_n(l(i))$ and return the computed result. Each simulated round of $\mathcal{A}$ will require $k_0$ communication rounds, thus the running time of $\mathcal{A}'$ is bounded by $k_0 g(n') \in o(\log^* n)$. At most $2k_0$ consecutive nodes will compute $b(i) = 0$, as $\mathcal{A}$ determines a DS, and by definition of $\mathcal{L}_{n'}^{k_0}$ at most $\sigma_{k_0}(n') - 1 \in \mathcal{O}(f(n')g(n')) \subset o(\log^*(n')) = o(\log^* n)$ consecutive nodes take the decision $b(i) = 1$. Hence $\mathcal{A}'$ is $o(\log^* n)$-alternating.

In the second case, no pair $k_0, n_0 \in \mathbb{N}$ as assumed in the first case exists. Thus, for any $k \in \mathbb{N}$ some $n \in \mathbb{N}$ exists for which we can construct a labeling of $R_n^k$ with at least $\frac{n}{2}$ many identifiers from disjoint sequences in $\mathcal{L}_n^k$. We line up these sequences one after another and label the remaining nodes in a way resulting in a valid labeling of $R_n^k$. Running $\mathcal{A}$ on such an instance will yield at least

$$\frac{n\sigma_k(n)}{2(\sigma_k(n) + 2kg(n))} \geq \frac{n}{6} \in \Omega(n)$$

nodes choosing $b(i) = 1$.

On the other hand, a minimum dominating set of $R_n^k$ has $\mathcal{O}(n/k)$ nodes. For $k \in \mathbb{N}$, define that $n_k$ is the minimum value of $n$ for which it is possible to construct a labeling of $R_n^k$ with $\frac{n}{2}$ identifiers from sequences in $\mathcal{L}_n^k$. Thus, we have a lower bound of

$$f(n_k) \in \Omega(k) \tag{11.1}$$

on the approximation ratio of $\mathcal{A}$.

As the approximation quality $f$ of $\mathcal{A}$ is sublinear, we conclude that $\lim_{k \to \infty} n_k = \infty$. Therefore, a minimum value $k(n)$ exists such that $n' := 2n < n_{k(n)}$. Consequently, we can define an injective relabeling function $\lambda_n : \{1, \dots, n\} \to \{1, \dots, n'\}$, such that no element of $\mathcal{L}_{n'}^{k(n)}$ lies completely in the image of $\lambda_n$. We define $\mathcal{A}'$ to be the algorithm operating on $R_n$, but returning the result of a simulated run of $\mathcal{A}$ on $R_{n'}^{k(n)}$, where we relabel all nodes $i \in V_n$ by $\lambda_n(l(i))$. By definition of $k(n)$ we have $n_{k(n)-1} \leq n'$. Together with (11.1) this yields

$$k(n) = (k(n) - 1) + 1 \in O(f(n_{k(n)-1}) + 1) \subseteq O(f(n')) = O(f(n)), \tag{11.2}$$

where the last step exploits the fact that $f$ grows asymptotically sublinearly. Hence we can estimate the running time of $\mathcal{A}'$ by $k(n)g(n') \in O(f(n)g(n))$, using that $g$ grows asymptotically sublinearly as well.

Since the simulated run of $\mathcal{A}$ yields a dominating set, at worst $2k(n) \in O(f(n)) \subseteq O(f(n)g(n))$ consecutive nodes may compute $b(v) = 0$. By the definitions of $\mathcal{L}_n^k$ and $\lambda_n$ at most $\sigma_{k(n)}(n') - 1 < \max\{f(n'), k(n)\}g(n') \in O(f(n)g(n))$ consecutive nodes may take the decision $b(i) = 1$. Thus $\mathcal{A}'$ is $o(\log^* n)$-alternating, as claimed. □

This result implies the lower bound, as the assumption that a good approximation ratio is possible leads to the contradiction that the ring could be 3-colored quickly.

**Theorem 11.7.** *No deterministic $f(n)$ MDS approximation on UDG's running in at most $g(n)$ rounds exists such that $f(n)g(n) \in o(\log^* n)$.*

*Proof.* Assuming the contrary, we may w.l.o.g. assume that $g(n) \geq 1$ for all $n \in \mathbb{N}$. Thus, we can combine Lemma 11.6 and Lemma 11.5 to construct an algorithm that 3-colors the ring in $o(\log^* n)$ rounds, contradicting Theorem 11.3. □

By the same technique, we can show that on the ring, an $f(n)$ MaxIS approximation that takes $g(n)$ rounds with $f(n)g(n) \in o(\log^* n)$ is impossible.

**Definition 11.8** (Maximum Independent Set Approximations)**.** *Given $f \in \mathbb{R}^+$, an IS $I$ is an $f$* MaxIS approximation*, if $f|I| \geq |M|$ for any MaxIS $M$. For $f : \mathbb{N} \to \mathbb{R}^+$, a deterministic $f$* approximation algorithm *for the MaxIS problem outputs on any graph of $n$ nodes an IS that is an $f(n)$ approximation.*

**Corollary 11.9.** *No deterministic $f(n)$ MaxIS approximation on the ring running in at most $g(n)$ rounds exists such that $f(n)g(n) \in o(\log^* n)$.*

*Proof.* See [65]. □

We remark that it is an open question whether a randomized algorithm can break this barrier for MDS approximations. For the MaxIS problem it is known that in planar graphs (and thus in particular on the ring), for any fixed constant $\varepsilon > 0$ a constant-time randomized algorithm can guarantee a $1 + \varepsilon$ approximation w.h.p. [25].

# Chapter 12

# Minimum Dominating Set Approximations in Graphs of Bounded Arboricity

*"Be greedy!" – An approach common in computer science, and maybe too common in other areas.*

This chapter presents two MDS approximation algorithms from [66] devised for graphs of small arboricity $A$. The first algorithm employs a forest decomposition, achieving a guaranteed approximation ratio of $\mathcal{O}(A^2)$ within $\mathcal{O}(\log n)$ rounds w.h.p. The second computes an $\mathcal{O}(A \log \Delta)$ approximation deterministically in $\mathcal{O}(\log \Delta)$ rounds. Both algorithms require small messages only.

## 12.1 Constant-Factor Approximation

In this section, we present an algorithm that computes a dominating set at most a factor of $\mathcal{O}(A^2)$ larger than optimum. After presenting the algorithm and its key ideas, we proceed with the formal proof of its properties.

### Algorithm

Our first algorithm is based on the following observations. Given an $f$-forest decomposition and an MDS $M$, the nodes can be partitioned into two sets. One set contains the nodes which are covered by a parent, the other contains the remaining nodes, which thus are themselves in $M$ or have a child in $M$. Since each dominating set node can cover at most $f$ parents, the latter set contains in total at most $(f+1)|M|$ many nodes. Even if all these covered nodes elect all of their parents into the dominating set, we have chosen at most $f(f+1)|M|$ nodes.

For the first set, we can exploit the fact that each node has at most $f$ parents in a more subtle manner. Covering the nodes in this set by parents only, we need to solve a special case of set cover where each element is part of at most $f$ sets. Such instances can be approximated well by a simple sequential greedy algorithm: Pick any element that is not yet covered and add *all* sets containing it; repeat this until no element remains. Since in each step we add at least one new set from an optimal solution, we get a factor $f$ approximation. This strategy can be parallelized by computing a maximal independent set in the graph where two nodes are adjacent exactly if they share a parent, as adding the parents of the nodes in an independent set in any order would be a feasible execution of the sequential greedy algorithm.

Putting these two observations together, first all parents of nodes from a maximal independent set in a helper graph are elected into the dominating set. In this helper graph, two nodes are adjacent if they share a parent. Afterwards, the remaining uncovered nodes have no parents, therefore it is uncritical with respect to the approximation ratio to select them all. Denoting for $v \in V$ by $P(v)$ the set of parents of $v$ in a given forest decomposition of $G$, this approach is summarized in Algorithm 12.1.

**Algorithm 12.1**: Parent Dominating Set

```
  input  : $f$-forest decomposition of $G$
  output: dominating set $D$
1 $H := \left(V, \left\{\{v,w\} \in \binom{V}{2} \mid P(v) \cap P(w) \neq \emptyset\right\}\right)$
2 Compute a maximal independent set $I$ with respect to $H$
3 $D := \bigcup_{v \in I} P(v)$
4 $D := D \cup (V \setminus \mathcal{N}_D^+)$
```

## Analysis

We need to bound the number of nodes that join the dominating set because they are elected by children.

**Lemma 12.1.** *In Line 3 of Algorithm 12.1, at most $f(f+2)|M|$ many nodes enter $D$, where $M$ denotes an MDS of $G$.*

*Proof.* Denote by $V_C \subseteq V$ the set of nodes that have a child in $M$ or are themselves in $M$. We have that $|V_C| \leq (f+1)|M|$, since no node has more than $f$ parents. Each such node adds at most $f$ parents to $D$ in Line 3 of the algorithm, i.e., in total at most $f(f+1)|M|$ many nodes join $D$ because they are elected by children in $I \cap V_C$.

Now consider the set of nodes $V_P \subseteq V$ that have at least one parent in $M$, in particular the nodes in $I \cap V_P$ that are also in the computed independent set. By the definition of $H$ and the fact that $I$ is an independent set, no node in $M$ can have two children in $I$. Thus, $|I \cap V_P| \leq |M|$. Since no node has more than $f$ parents, we conclude that at most $f|M|$ many nodes join $|D|$ because they are elected into the set by a child in $I \cap V_P$.

Finally, observe that since $M$ is a dominating set, we have that $V_C \cup V_P = V$ and thus

$$|D| \leq f|I \cap V_C| + f|I \cap V_P| \leq f(f+1)|M| + f|M| = f(f+2)|M|,$$

concluding the proof. □

The approximation ratio of the algorithm now follows easily.

**Theorem 12.2.** *Algorithm 12.1 outputs a dominating set $D$ containing at most $(f^2+3f+1)|M|$ nodes, where $M$ is an optimum solution.*

*Proof.* By Lemma 12.1, at most $f(f+2)|M|$ nodes enter $D$ in Line 3 of the algorithm. Since $I$ is an MIS in $H$, all nodes that have a parent are adjacent to at least one node in $D$ after Line 3. Hence, the nodes selected in Line 4 must be covered by a child in $M$ or themselves be in $M$. As no node has more than $f$ parents, thus in Line 4 at most $(f+1)|M|$ many nodes join $D$. Altogether, at most $(f^2+3f+1)|M|$ many nodes may end up in $D$ as claimed. □

Employing known algorithms for computing an $\mathcal{O}(G(A))$-forest decomposition and an MIS, we can construct a distributed MDS approximation algorithm.

**Corollary 12.3.** *In any graph $G$, a factor $\mathcal{O}(A(G)^2)$ approximation to an MDS can be computed distributedly in $\mathcal{O}(\log n)$ rounds w.h.p., provided that nodes know a polynomial upper bound on $n$, or a linear upper bound on $A(G)$. In particular, on graphs of bounded arboricity a constant-factor approximation can be obtained in $\mathcal{O}(\log n)$ rounds w.h.p. This can be accomplished with messages of size $\mathcal{O}(\log n)$.*

*Proof.* We run Algorithm 12.1 in a distributed fashion. To see that this is possible, observe that $(i)$ nodes need to know whether a neighbor is a parent or a child only, $(ii)$ that $H$ can be constructed locally in two rounds and $(iii)$ a synchronous round in $H$ can be simulated by two rounds in $G$. Thus, we may simply pick distributed algorithms to compute a forest decomposition of $G$ and an MIS and plug them together to obtain a distributed variant of Algorithm 12.1.

For the forest decomposition, we employ the algorithm from [8], yielding a decomposition into $\mathcal{O}(A(G))$ forests in $\mathcal{O}(\log n)$ rounds; this algorithm is the one that requires the bound on $n$ or $A(G)$, respectively, that is asked for in the preliminaries of the corollary. An MIS can be computed in $\mathcal{O}(\log n)$ rounds w.h.p. by well-known algorithms [2, 43, 73], or a more recent similar technique [80]. In total the algorithm requires $\mathcal{O}(\log n)$ rounds w.h.p. and according to Theorem 12.2 the approximation guarantee is $\mathcal{O}(A(G)^2)$.

Regarding the message size, the algorithm to compute a forest decomposition requires messages of $\mathcal{O}(\log n)$ bits. Thus, we need to check that we do not require large messages because we compute an MIS on $H$. Formulated abstractly, the algorithm from [80] breaks symmetry by making each node still eligible for entering the IS choosing a random value in each round and permitting it to join the IS if its value is a local minimum. This concept can for instance be realized by taking $\mathcal{O}(\log n)$ random bits as encoding of some number and comparing it to neighbors. The respective values will differ w.h.p. This approach can be emulated using messages of size $\mathcal{O}(\log n)$ in $G$: Nodes send their random values to all parents in the forest decomposition, which then forward the smallest values only to their children.[1] □

We remark that the same approach can be used by a centralized algorithm in order to compute an $\mathcal{O}(A(G)^2)$ approximation within $\mathcal{O}(A(G)n)$ steps. A sequential algorithm does not need to incur an overhead of $\mathcal{O}(\log n)$, as a forest decomposition can be determined in linear time and finding an MIS becomes trivial.

**Corollary 12.4.** *Deterministically, on any graph $G$ an $\mathcal{O}(A(G)^2)$ MDS approximation can be computed in $\mathcal{O}(|E| + n) \subseteq \mathcal{O}(A(G)n)$ centralized steps.*

*Proof.* See [66]. □

## 12.2 Uniform Deterministic Algorithm

Algorithm 12.1 might be unsatisfactory with regard to several aspects. Its running time is logarithmic in $n$ even if the maximum degree $\Delta$ is small. This cannot be improved upon by any approach that utilizes a forest decomposition, as a lower bound of $\Omega(\log n / \log f)$ is known on the time to compute a forest decomposition into $f$ forests [8]. The algorithm is not uniform, as it necessitates global knowledge of a bound on $A(G)$ or $n$.

[1] If (an upper bound on) $n$ is not known, one can start with constantly many bits and double the number of used bits in each round where two nodes pick the same value. This will not slow down the algorithm significantly and bound message size by $\mathcal{O}(\log n)$ w.h.p.

Moreover, the algorithm requires randomization in order to compute an MIS quickly. Considering deterministic algorithms, one might pose the question how much initial symmetry breaking information needs to be provided to the nodes. While randomized algorithms may generate unique identifiers of size $\mathcal{O}(\log n)$ in constant time w.h.p., many deterministic algorithms assume them to be given as input. Milder assumptions are the ability to distinguish neighbors by means of a port numbering and/or an initially given orientation of the edges.

In this section, we show that a uniform, deterministic algorithm exists that requires a port numbering only, yet achieves a running time of $\mathcal{O}(\log \Delta)$ and a good approximation ratio. The size of the computed dominating set is bounded linearly in the product of the arboricity $A(G)$ of the graph and the logarithm of the maximum degree $\Delta$.

## Algorithm

The basic idea of Algorithm Greedy-by-Degree (Algorithm 12.2) is that it is always feasible to choose nodes of high *residual degree* simultaneously, i.e., all the nodes that cover up to a constant factor as many nodes as the one covering the most uncovered nodes.

**Definition 12.5** (Residual Degree)**.** *Given a set $D \subseteq V$, the residual degree of node $v \in V$ with respect to $D$ is $\bar{\delta}_v := |\mathcal{N}_v^+ \setminus \mathcal{N}_D^+|$.*

This permits to obtain strong approximation guarantees without the structural information provided by knowing $A(G)$ or a forest decomposition; the mere fact that the graph must be "locally sparse" enforces that if many nodes are elected into the set, also the dominating set must be large. A difficulty arising from this approach is that nodes are not aware of the current maximum residual degree in the graph. Hence, every node checks whether there is a node in its 2-hop neighborhood having a residual degree larger by a factor two. If not, the respective nodes may join the dominating set (even if their degree is not large from a global perspective), implying that the maximum residual degree drops by a factor of two in a constant number of rounds.

A second problem occurs once residual degrees become small. In fact, it may happen that a huge number of already covered nodes can each cover the same small set of $A(G) - 1$ nodes. For this reason, it is mandatory to ensure that not more nodes join the dominating set than actually need to be covered. To this end, nodes that still need to be covered elect one of their neighbors (if any) that are feasible according to the criterion of (locally) large residual degree explained above. This scheme is described in Algorithm 12.2.

Note that nodes never leave $D$ once they entered it. Thus, nodes may terminate based on local knowledge only when executing the algorithm, as

**Algorithm 12.2**: Greedy-by-Degree.

```
   output: dominating set $D$
1  $D := \emptyset$
2  while $V \neq \mathcal{N}_D^+$ do
3      $C := \emptyset$ // candidate set
4      for $v \in V$ in parallel do
5          $\bar{\delta}_v := |\mathcal{N}_v^+ \setminus \mathcal{N}_D^+|$ // residual degree
6          $\Delta_v := \max_{w \in \mathcal{N}_v^+}\{\bar{\delta}_w\}$ // maximum within one hop
7          $\Delta_v := \max_{w \in \mathcal{N}_v^+}\{\Delta_w\}$ // maximum within two hops
8          if $\lceil \log \bar{\delta}_v \rceil \geq \lceil \log \Delta_v \rceil$ then
9              $C := C \cup \{v\}$
10         end
11         if $v \in \mathcal{N}_C^+ \setminus \mathcal{N}_D^+$ then
12             choose any $w \in C \cap \mathcal{N}_v^+$ // e.g. smallest port number
13             $D := D \cup \{w\}$ // uncovered nodes select a candidate
14         end
15     end
16 end
```

they can cease executing the algorithm as soon as $\bar{\delta}_v = 0$, i.e., their entire inclusive neighborhood is covered by $D$. Moreover, it can easily be verified that one iteration of the loop can be executed within six rounds by a local algorithm that relies on port numbers only.

## Analysis

In the sequel, when we talk of a *phase* of Algorithm 12.2, we refer to a complete execution of the while loop. We start by proving that not too many nodes with small residual degrees enter $D$.

**Lemma 12.6.** *Denote by $M$ an MDS of $G$. During the execution of Algorithm 12.2, in total at most $16A(G)|M|$ nodes join $D$ in Line 13 of the algorithm after computing $\bar{\delta}_v \leq 8A(G)$ in Line 5 of the same phase.*

*Proof.* Fix a phase of the algorithm. Consider the set $S$ consisting of all nodes $v \in V$ that become covered in some phase by some node $w \in \mathcal{N}_v^+$ that computes $\bar{\delta}_w \leq 8A(G)$ and joins $D$. As according to Line 8 nodes join $D$ subject to the condition that residual degrees throughout their 2-hop neighborhoods are less than twice as large as their own, no node $m \in M$ can cover more than $16A(G)$ nodes from $S$. Hence, $|S| \leq 16A(G)|M|$. Because of the rule that a node needs to be elected by a covered node in order to enter $D$, this is also a bound on the number of nodes joining $D$ in a phase when they have residual degree at most $8A(G)$. □

Next, we show that in each phase, at most a constant factor more nodes of large residual degree are chosen than are in an MDS.

**Lemma 12.7.** *If $M$ is an MDS, in each phase of Algorithm 12.2 at most $16A(G)|M|$ nodes $v \in V$ that compute $\bar{\delta}_v > 8A(G)$ in Line 5 join $D$ in Line 13.*

*Proof.* Fix some phase of the algorithm and denote by $D'$ the set of nodes $v \in V$ joining $D$ in Line 13 of this phase after computing $\bar{\delta}_v > 8A(G)$. Define $V'$ to be the set of nodes that had not been covered at the beginning of the phase. Define for $i \in \{0, \ldots, \lceil \log n \rceil\}$ that

$$\begin{aligned}
M_i &:= \{v \in M \mid \bar{\delta}_v \in (2^{i-1}, 2^i]\}\\
V_i &:= \left\{v \in V' \;\middle|\; \max_{w \in \mathcal{N}_v^+}\{\bar{\delta}_w\} \in (2^{i-1}, 2^i]\right\}\\
D_i &:= \{v \in D' \mid \bar{\delta}_v \in (2^{i-1}, 2^i]\}.
\end{aligned}$$

Note that $\bigcup_{i=\lceil \log 8A(G)\rceil}^{\lceil \log n \rceil} D_i = D'$.

Consider any $j \in \{\lceil \log 8A(G)\rceil, \ldots, \lceil \log n \rceil\}$. By definition, nodes in $V_j$ may be covered by nodes from $M_i$ for $i \le j$ only. Thus,

$$\sum_{i=0}^{j} 2^i |M_i| \ge |V_j|.$$

Nodes $v \in D_j$ cover at least $2^{j-1}+1$ nodes from the set $\bigcup_{i \in \{j, \ldots, \lceil \log n \rceil\}} V_i$, as by definition they have no neighbors in $V_i$ for $i < j$. On the other hand, Lines 5 to 8 of the algorithm impose that these nodes must not have any neighbors of residual degree larger than $2^{\lceil \log \bar{\delta}_v \rceil} = 2^j$, i.e., these nodes cannot be in a set $V_i$ for $i > j$. Hence, each node $v \in D_j$ has at least $2^{j-1}$ neighbors in $V_j$. This observation implies that the subgraph induced by $D_j \cup V_j$ has at least $2^{j-2}|D_j| \ge 2A(G)|D_j|$ edges. On the other hand, by definition of the arboricity, this subgraph has fewer than $A(G)(|D_j| + |V_j|)$ edges. It follows that

$$|D_j| < \frac{A(G)|V_j|}{2^{j-2} - A(G)} \le 2^{3-j} A(G)|V_j| \le 2^{3-j} A(G) \sum_{i=0}^{j} 2^i |M_i|.$$

We conclude that

$$\begin{aligned}
|D'| &= \sum_{j=\lceil \log 8A(G)\rceil}^{\lceil \log n\rceil} |D_j| \\
&\leq \sum_{j=\lceil \log 8A(G)\rceil}^{\lceil \log n\rceil} 2^{3-j}A(G)\sum_{i=0}^{j} 2^i|M_i| \\
&\leq 8A(G)\sum_{j=0}^{\lceil \log n\rceil}\sum_{i=0}^{j} 2^{i-j}|M_i| \\
&< 8A(G)\sum_{i=0}^{\lceil \log n\rceil}\sum_{j=i}^{\infty} 2^{i-j}|M_i| \\
&= 16A(G)\sum_{i=0}^{\lceil \log n\rceil} |M_i| \\
&\leq 16A(G)|M|,
\end{aligned}$$

as claimed. □

We now can bound the approximation quality of the algorithm.

**Theorem 12.8.** *Algorithm 12.2 terminates within* $6\lceil\log(\Delta+1)\rceil$ *rounds and outputs a dominating set that is at most a factor* $16A(G)\log\Delta$ *larger than optimum. The message size can be bounded by* $\mathcal{O}(\log\log\Delta)$.

*Proof.* We first examine the running time of the algorithm. Denote by $\Delta(i)$ the maximum residual degree after the $i^{th}$ phase, i.e., $\Delta(0) = \Delta + 1$ (as a node also covers itself). As observed earlier, each phase of Algorithm 12.2 takes six rounds. Because all nodes $v$ computing a $\bar{\delta}_v$ satisfying $\lceil\log\bar{\delta}_v\rceil = \lceil\log\Delta(i)\rceil$ join $C$ in phase $i$ and any node in $\mathcal{N}_C^+$ becomes covered, we have that $\lceil\log\Delta(i+1)\rceil \leq \lceil\log\Delta(i)\rceil - 1$ for all phases $i$. Since the algorithm terminates at the end of the subsequent phase once $\Delta(i) \leq 2$, in total at most $\lceil\log\Delta(0)\rceil = \lceil\log(\Delta+1)\rceil$ phases are required.

Having established the bound on the running time of the algorithm, its approximation ratio directly follows[2] by applying Lemmas 12.6 and 12.7. The bound on the message size follows from the observation that in each phase nodes need to exchange residual degrees rounded to powers of two and a constant number of binary values only. □

Like it is possible for the MDS approximation algorithm for general graphs from [56], we can sacrifice accuracy in order to speed up the computation.

[2]Note that in the last three phases the maximum degree is at most $8 \leq 8A(G)$.

**Corollary 12.9.** *For any integer $\alpha \geq 2$, Algorithm 12.2 can be modified such that it has a running time of $\mathcal{O}(\log_\alpha \Delta)$ and approximation ratio $\mathcal{O}(A(G)\alpha \log_\alpha \Delta)$. The size of messages becomes $\mathcal{O}(\log \log_\alpha \Delta)$ with this modification.*

*Proof.* We simply change the base of the logarithms in Line 8 of the algorithm, i.e., instead of rounding residual degrees to integer powers of two, we round to integer powers of $\alpha$. Naturally, this affects the approximation guarantees linearly. In the proof of Lemma 12.7, we just replace the respective powers of two by powers of $\alpha$ as well, yielding a bound of $\mathcal{O}(A(G)\alpha \log_\alpha \Delta)$ on the approximation ratio by the same reasoning as in Theorem 12.8. Similarly, the bound on the message size becomes $\mathcal{O}(\log \log_\alpha \Delta)$. □

If it was not for the computation of an MIS, we could speed up Algorithm 12.1 in almost the same manner (accepting a forest decomposition into a larger number of forests). However, the constructed helper graph is of bounded independence, but not arboricity or growth. For this graph class currently no distributed algorithm computing an MIS in time $o(\log n)$ is known.

Finally, we would like to mention that if nodes know $A(G)$ (or a reasonable upper bound), a port numbering is not required anymore. In this case, nodes will join $D$ without the necessity of being elected by a neighbor, however only if the prerequisite $\bar{\delta}_v > 8A(G)$ is satisfied. To complete the dominating set, uncovered nodes may join $D$ independently of $\bar{\delta}_v$ once their neighborhood contains no more nodes of residual degree larger than $8A(G)$. It is not hard to see that with this modification, essentially the same analysis as for Algorithm 12.2 applies, both with regard to time complexity and approximation ratio.

# Chapter 13

# Minimum Dominating Set Approximations in Planar Graphs

*"Well, it's a constant." – Jukka Suomela on the approximation ratio of the algorithm presented in this chapter.*

In this chapter, which is based on [62], we introduce an algorithm computing a constant approximation of a minimum dominating set in planar graphs in constant time.[1] Assuming maximum degree $\Delta$ and identifiers of size $\mathcal{O}(\log n)$, the algorithm makes use of messages of size $\mathcal{O}(\Delta \log n)$. As planar graphs exhibit unbounded degree, the algorithm is thus not suitable for practice. Moreover, the constant in the approximation ratio is 130, i.e., there is a large gap to the lower bound of $5 - \varepsilon$ (for any constant $\varepsilon > 0$). Nevertheless, we demonstrate that in planar graphs in principle it is feasible to obtain a constant MDS approximation in a constant number of distributed rounds.

## 13.1 Algorithm

The key idea of the algorithm is to exploit planarity in two ways. On the one hand, planar graphs have arboricity three, i.e., the number of edges of any subgraph is linear in its number of nodes. What is more, as planarity is preserved under taking minors, so does any minor of the graph. On the other hand, in a planar graph circles are barriers separating parts of the graph from

[1] Note that the original paper [61] contained an error and the stated algorithm does not compute a constant MDS approximation. Moreover, the proof of the algorithm from [25] is incomplete.

**Algorithm 13.1**: MDS Approximation in Planar Graphs

```
   output: DS $D$ of $G$
1  $D := \emptyset$
2  for $v \in V$ in parallel do
3      if $\nexists A \subseteq \mathcal{N}_v^{(2)} \setminus \{v\}$ such that $\mathcal{N}_v \subseteq \mathcal{N}_A^+$ and $|A| \leq 6$ then
4          $D := D \cup \{v\}$
5      end
6  end
7  for $v \in V$ in parallel do
8      $\bar{\delta}_v := |\mathcal{N}_v^+ \setminus \mathcal{N}_D^+|$ // residual degree
9      if $v \in V \setminus \mathcal{N}_D^+$ then
10         $\Delta_v := \max_{w \in \mathcal{N}_v^+}\{\bar{\delta}_w\}$ // maximum within one hop
11         choose any $d(v) \in \{w \in \mathcal{N}_v^+ \mid \bar{\delta}_w = \Delta_v\}$
12         $D := D \cup \{d(v)\}$
13     end
14 end
```

others; any node enclosed in a circle cannot cover nodes on the outside. This is a very strong structural property enforcing that dominating sets are either large or exhibit a simple structure. It will become clear in the analysis how these properties are utilized by the algorithm.

The algorithm consists of two main steps. In the first step all nodes check whether their neighborhood can be covered by six or less other nodes. After learning about their two-hop neighborhood in two rounds, nodes can decide this locally by means of a polynomial-time algorithm.[2] If this is not the case, they join the (future) dominating set. In the second step, any node that is not yet covered elects a neighbor of maximal residual degree (i.e., one that covers the most uncovered nodes, see Definition 12.5) into the set. Algorithm 13.1 summarizes this scheme.

## 13.2 Analysis

As evident from the description of the algorithm, it can be executed in six rounds and, due to the second step, computes a dominating set. Therefore, we need to bound the number of nodes selected in each step in terms of the size of a minimum dominating set $M$ of the planar graph $G$. For the purpose of our analysis, we fix some MDS $M$ of $G$. By $D_1$ and $D_2$ we denote the sets of

[2]Trivially, one can try all combinations of six nodes. Note, however, that planarity permits more efficient solutions.

nodes that enter $D$ in the first and second step of the algorithm, respectively. Moreover, we denote neighborhoods in a graph $H \neq G$ by $\mathcal{N}_v(H)$, $\mathcal{N}_A^+(H)$, etc.

We begin by bounding the number of nodes in $D_1 \setminus M$ after the first step.

**Lemma 13.1.** $|D_1 \setminus M| < 3|M|$.

*Proof.* We construct the following subgraph $H = (V_H, E_H)$ of $G$ (see Figure 13.1).

- Set $V_H := \mathcal{N}^+_{D_1 \setminus M} \cup M$ and $E_H := \emptyset$.
- Add all edges with an endpoint in $D_1 \setminus M$ to $E_H$.
- Add a minimal subset of edges from $E$ to $E_H$ such that $V_H = \mathcal{N}_M^+(H)$, i.e., $M$ is a DS in $H$.

Thus, each node $v \in V_H \setminus (D_1 \cup M)$ has exactly one neighbor $m \in M$, as we added a minimal number of edges for $M$ to cover $V_H$. For all such nodes $v$, we contract the edge $\{v, m\}$, where we identify the resulting node with $m$. In other words, the star subgraph of $H$ induced by $\mathcal{N}_m^+(H) \setminus D_1$ is collapsed into $m$. By Lemma 2.24, the resulting minor $\bar{H} = (V_{\bar{H}}, E_{\bar{H}})$ of $G$ satisfies that $|E_{\bar{H}}| < 3|V_{\bar{H}}|$. Due to the same lemma, the subgraph of $\bar{H}$ induced by $D_1 \setminus M$ has fewer than $3|D_1 \setminus M|$ edges. As the neighborhood of a node from $D_1 \setminus M \subset V_{\bar{H}}$ cannot be covered by fewer than seven nodes, the performed edge contractions did not reduce the degree of such a node below seven.

Altogether, we get that

$$
\begin{aligned}
& 7|D_1 \setminus M| - 3|D_1 \setminus M| \\
< \; & \sum_{d \in D_1 \setminus M} \delta_d(\bar{H}) - \left|\left\{\{d, d'\} \in E_{\bar{H}} \,\middle|\, d, d' \in D_1 \setminus M\right\}\right| \\
\leq \; & |E_{\bar{H}}| \\
< \; & 3|V_{\bar{H}}| \\
\leq \; & 3(|D_1 \setminus M| + |M|),
\end{aligned}
$$

which can be rearranged to yield the claimed bound. □

To bound the number of nodes $|D_2|$ that is chosen in the second step of the algorithm, more effort is required. We consider the following subgraph of $G$.

**Definition 13.2.** *We define $H = (V_H, E_H)$ to be the subgraph of $G$ obtained from the following construction.*

- *Set $V_H := \emptyset$ and $E_H := \emptyset$.*

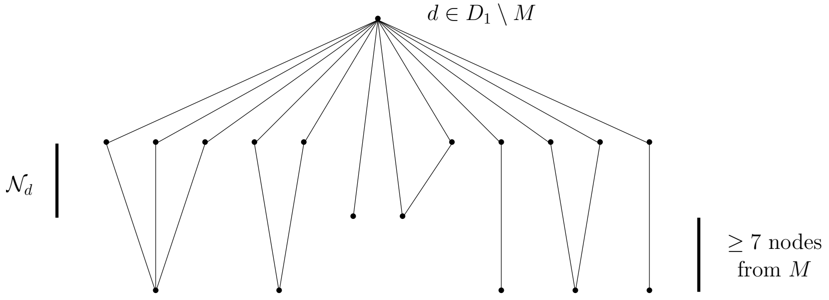

Figure 13.1: Part of the subgraph constructed in Lemma 13.1.

- *For each node $d \in D_2$ for which this is possible, add one node $v \in V \setminus M$ to $V_H$ such that $d = d(v)$ in Line 11 of the algorithm.*
- *Add $M \setminus D_1$ to $V_H$ and a minimal number of edges to $E_H$ such that $\mathcal{N}^+_{M \setminus D_1}(H) = V_H$, i.e., $M \setminus D_1$ covers the nodes added to $H$ so far (this is possible as only nodes from $V \setminus \mathcal{N}^+_{D_1}$ elect nodes into $D_2$).*
- *For each $m \in M \setminus D_1$, add a minimal number of nodes and edges to $H$ such that there is a set $C_m \subseteq V_H \setminus \{m\}$ of minimal size satisfying $\mathcal{N}_m(H) \subseteq \mathcal{N}^+_{C_m}(H)$, i.e., $C_m$ covers $m$'s neighbors in $H$. We define that $C := \cup_{m \in M \setminus D_1} C_m$.*
- *Remove all nodes $v \in V_H \setminus (C \cup M)$ for which $d(v) \in M \cup C$.*
- *For each $m \in M \setminus D_1$, remove all edges to $C_m$.*

*See Figure 13.2 for an illustration.*

In order to derive our bound on $|D_2|$, we consider a special case first.

**Lemma 13.3.** *Assume that for each node $m \in M \setminus D_1$ it holds that*

*(i) no node $m' \in M \cap C_m$ covers more than seven nodes in $\mathcal{N}_m(H)$ and*

*(ii) no node $v \in C_m \setminus M$ covers more than four nodes in $\mathcal{N}_m(H)$.*

*Then it holds that $|D_2| < 98|M|$.*

*Proof.* Denote by $A_1 \subseteq V_H \setminus (M \cup C)$ the nodes in $V_H$ that elect others into $D_2$ and have two neighbors in $M$, i.e., when we added $C$ to $V_H$, they became covered by a node in $M \cap C$. Analogously, denote by $A_2 \subseteq V_H \setminus (M \cup C)$ the set of electing nodes for which the neighbor in $C$ is not in $M$. Observe that

$A := A_1 \cup A_2 = V_H \setminus (M \cup C)$ and $A_1 \cap A_2 = \emptyset$. Moreover, we claim that $|A| \geq |D_2| - 14|M|$. To see this, recall that in the first step of the construction of $H$, we choose for each element of $|D_2|$ that is not elected by elements of $M$ only one voting node $v$, i.e., at least $|D_2| - |M|$ nodes in total. In the second last step of the construction, we remove $v$ if $d(v) \in \{m\} \cup C_m$ for some $m \in M \setminus D_1$. As $m \in M \setminus D_1$, its neighborhood can be covered by six or less nodes from $V \setminus \{m\}$. Therefore $|C_m| \leq 6$ for all $M \setminus D_1$ and we remove in total at most $7|M|$ nodes in the second last step. Finally, in the last step we cut off at most $|C| \leq 6|M|$ voting nodes from their dominators in $M \setminus D_1$. The definition of $A$ explicitly excludes these nodes, hence $|A| \geq |D_2| - 14|M|$.

We contract all edges from nodes $a \in A$ to the respective nodes $m \in M \setminus D_1$ covering them we added in the third step of the construction of $H$. Denote the resulting minor of $G$ by $\bar{H} = (V_{\bar{H}}, E_{\bar{H}})$. For every seven nodes in $A_1$, there must be a pair of nodes $m, m' \in M \setminus D_1$ such that $m \in C_{m'}$ and vice versa, as by assumption no such pair shares more than seven neighbors. Thus, for every seven nodes in $A_1$, we have two nodes less in $V_{\bar{H}}$ than the upper bound of $|V_{\bar{H}}| \leq |M| + |C| \leq 7|M|$. By Lemma 2.24, $\bar{H}$ thus has fewer than

$$3|V_{\bar{H}}| \leq 3|M \cup C| \leq 3|M| + 3\left(6|M| - \frac{2|A_1|}{7}\right) = 21|M| - \frac{6|A_1|}{7}$$

edges.

On the other hand,

$$|E_{\bar{H}}| \geq \frac{|A_1|}{7} + \frac{|A_2|}{4},$$

as by assumption each pair of nodes from $M$ may share at most seven neighbors in $A_1$ and pairs of nodes $m \in M \setminus D_1$, $v \in C_m \setminus M$ share at most four neighbors. We conclude that

$$|A_2| < 84|M| - 4|A_1|$$

and therefore

$$|D_2| \leq |A_1| + |A_2| + 14|M| < 98|M| - 3|A_1| \leq 98|M|. \qquad \square$$

In order to complete our analysis, we need to cope with the case that a node $m \in M \setminus D_1$ and an element of $C_m$ share many neighbors. In a planar graph, this results in a considerable number of nested circles which separate their interior from their outside. This necessitates that nodes from the optimal solution $M$ are enclosed that we may use to compensate for the increased number of nodes in $A$ in comparison to the special case from Lemma 13.3.

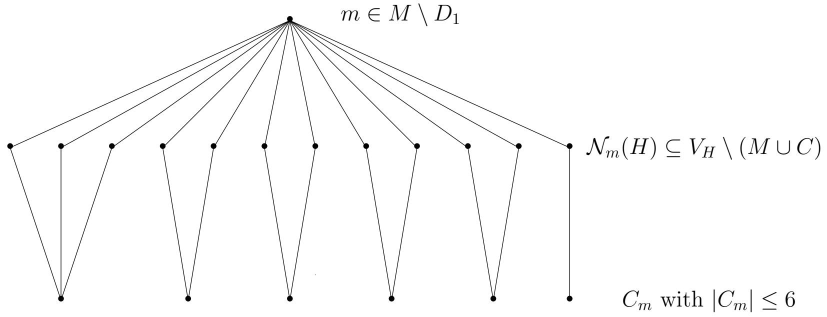
$m \in M \setminus D_1$
$\mathcal{N}_m(H) \subseteq V_H \setminus (M \cup C)$
$C_m$ with $|C_m| \leq 6$

Figure 13.2: Part of the subgraph $H$ from Definition 13.2.

**Lemma 13.4.** *Suppose the subgraph $H$ from Definition 13.2 violates condition $(i)$ or $(ii)$ from Lemma 13.3. Fix a planar embedding of $G$ and consider either*

*(i) nodes $m \in M \setminus D_1$ and $v \in M \cap C_m$ with $|\mathcal{N}_m(H) \cap \mathcal{N}_v(H)| \geq 8$ or*

*(ii) nodes $m \in M \setminus D_1$ and $v \in C_m \setminus M$ with $|\mathcal{N}_m(H) \cap \mathcal{N}_v(H)| \geq 5$.*

*Then the outmost circle formed by $m$, $v$, and two of their common neighbors in $H$ must enclose some node $m' \in M$ (with respect to the embedding).*

*Proof.* Set $\tilde{A} := \mathcal{N}_m(H) \cap \mathcal{N}_v(H)$. Consider case $(i)$ first and assume for contradiction that there is no node from $M$ enclosed in the outmost circle. W.l.o.g., we may assume that $|\tilde{A}| = 8$ (otherwise we simply ignore some nodes from $\tilde{A}$). There are four nodes from $\tilde{A}$ that are enclosed by two nested circles consisting of $v$, $m$, and the four nodes that are the outer nodes from $\tilde{A}$ according the embedding (see Figure 13.3). Recall that by the second last step of the construction of $H$ nodes $a \in \tilde{A}$ satisfy that $d(a) \notin \{m, v\} \subseteq M$. Therefore, these enclosed nodes elected (distinct) nodes into $D_2$ that are enclosed by the outmost circle. As the electing nodes $a \in \tilde{A}$ are connected to $m$ and $v$, by Line 11 of the Algorithm the nodes $d(a)$ they elected must have at least residual degree $\bar{\delta}_{d(a)} \geq \max\{\bar{\delta}_v, \bar{\delta}_m\}$. In other words, they cover as least as many nodes from $V \setminus \mathcal{N}^+_{D_1}$ as both $m$ and $v$.

Denote by $\ell$ the number of enclosed nodes from $G$ that are neither in $\tilde{A} \subseteq V \setminus \mathcal{N}^+_{D_1}$ nor already covered by $D_1$. We thus have a subgraph $S = (V_S, E_S)$ of $G$ that has $|V_S| = l + |\tilde{A}| + |\{v, m\}| = \ell + 10$ nodes and

$$\begin{aligned} |E_S| &\geq |\mathcal{N}_v(S)| + |\mathcal{N}_m(S)| + 4\max\{|\mathcal{N}_v(S)|, |\mathcal{N}_m(S)|\} - 18 \\ &\geq 3(|\mathcal{N}_v(S)| + |\mathcal{N}_m(S)| - 6) \end{aligned}$$

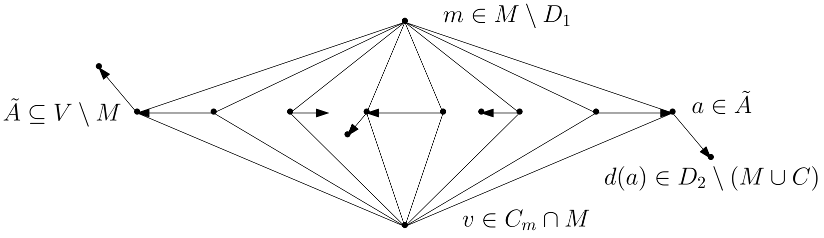
$m \in M \setminus D_1$
$\tilde{A} \subseteq V \setminus M$
$a \in \tilde{A}$
$d(a) \in D_2 \setminus (M \cup C)$
$v \in C_m \cap M$

Figure 13.3: Example of a subgraph considered in the first case of the proof of Lemma 13.4. While the choices $d(a)$ of the two leftmost and rightmost nodes $a \in \tilde{A}$ may have large degrees because of nodes outside the outer circle, the choices of the four inner nodes must have many neighbors that are not covered by $D_1$ on or inside the outer circle.

edges, where we subtracted 18 because (*i*) no edge is required for one of the four nodes to cover itself, (*ii*) we might have counted $\binom{4}{2} = 6$ edges between pairs of the four considered nodes $d(a) \in D_2$ twice, and (*iii*) we might have counted up to 8 edges between these four nodes and $\{v, m\}$ twice. As we made sure that $\tilde{A} \cap M = \emptyset$ by adding nodes from $V \setminus M$ only to $V_H$ in the second construction step in Definition 13.2, the assumption that no other node from $M$ is enclosed by the outmost circle implies that everything inside is covered by $\{v, m\}$. Therefore, it holds that

$$|\mathcal{N}_v(S)| + |\mathcal{N}_m(S)| \geq 2|\tilde{A}| + \ell = \ell + 16.$$

However, Lemma 2.24 lower bounds $|V_S|$ in terms of $|E_S|$, yielding

$$3(\ell + 10) = 3|V_S| > |E_S| \geq 3(|\mathcal{N}_v(S)| + |\mathcal{N}_m(S)| - 6) \geq 3(\ell + 10),$$

a contradiction.

Case (*ii*) is treated similarly, but it is much simpler. This time, the assumption that no node from $M$ is enclosed by the outmost circle implies that all the nodes inside must be covered by $m$ alone, as $M$ is a DS. Since $v$ and $m$ are connected via the (at least) five nodes in $\tilde{A}$, for the node $d(a) \notin \{m, v\}$ elected into $D_2$ by the innermost node $a \in \tilde{A}$, it must hold that $\mathcal{N}^+_{d(a)} \setminus \mathcal{N}^+_m \subseteq \{v\}$ (see Figure 13.4). However, there are at least two nodes in $\tilde{A} \subseteq V \setminus \mathcal{N}^+_{D_1}$ that are not connected to $d(a)$, i.e., we get the contradiction that $a$ would have preferred $m$ over $d(a)$ in Line 11 of the algorithm. □

Next, we repeatedly delete nodes from $H$ until eventually the preconditions of Lemma 13.3 are met. Arguing as in the proof of Lemma 13.4, we can

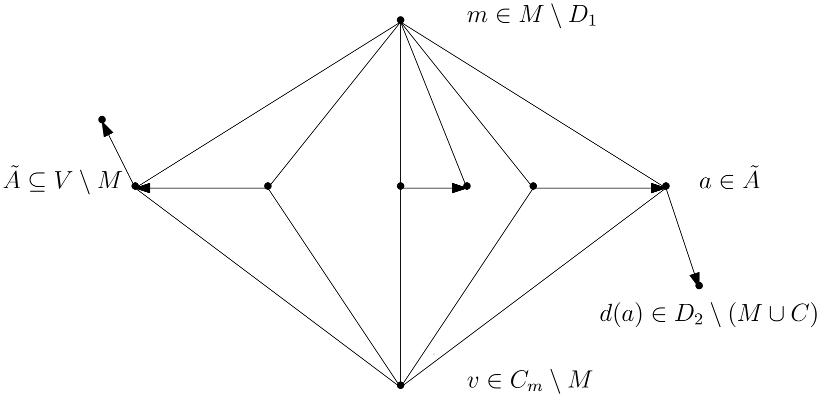
$m \in M \setminus D_1$
$\tilde{A} \subseteq V \setminus M$
$a \in \tilde{A}$
$d(a) \in D_2 \setminus (M \cup C)$
$v \in C_m \setminus M$

Figure 13.4: Example of a subgraph considered in the second case of the proof of Lemma 13.4. Supposing there is no other node $m' \in M$ inside the outer circle, apart from $v$ all neighbors of the node chosen by the innermost node from $\tilde{A}$ must also be neighbors of $m$.

account for deleted nodes by allocating them to enclosed nodes from $M \cup C$. Doing this carefully, we can make sure that no nodes from $M \cup C$ need to compensate for more than four deleted nodes.

**Corollary 13.5.** $|D_2| < 126|M|$.

*Proof.* Fix an embedding of $G$ and thus of all its subgraphs. We will argue with respect to this embedding only. We use the notation from the proof of Lemma 13.3. Starting from $H$, we iteratively delete nodes from $A$ until we obtain a subgraph $H'$ satisfying the prerequisites of the lemma. Assume that $H' := H$ violates one of the preconditions of Lemma 13.3. No matter which of the conditions $(i)$ and $(ii)$ from Lemma 13.3 is violated, we choose respective nodes $m \in M \setminus D_1$ and $v \in C_m$ satisfying precondition $(i)$ or $(ii)$ of Lemma 13.4 such that the smallest circle formed by $m$, $v$, and $a_1, a_2 \in \tilde{A} := \mathcal{N}_v^+(H') \cap \mathcal{N}_m(H')$ enclosing an element $m' \in M$ has minimal area. We delete the two elements from $\tilde{A} \subseteq A$ participating in the circle. Since the area of the circle is minimal, there is no third element from $\tilde{A}$ enclosed in the circle.

We repeat this process until $H'$ satisfies the preconditions of Lemma 13.3. We claim that we can account for deleted nodes in terms of nodes from $M \cup C$ in a way such that no element of $M \cup C$ needs to compensate for more than four deleted nodes. Whenever we delete a pair of nodes, we count a node from $M \cup C$ enclosed by the respective circle that has not yet been counted twice.

We need to show that this is indeed always possible. To see this, observe that the minimality of the enclosed area of a chosen circle $X$ together with the planarity of $G$ ensures that any subsequent circle $X'$ either encloses this circle or its enclosed area is disjoint from the one of $X$. In the latter case, we obviously must find a different node from $M \cup C$ enclosed in $X'$ than the one we used when deleting nodes from $X$. Hence, we need to examine the case when there are three nested circles $X_1$, $X_2$, and $X_3$ that occur in the construction. If the nodes $m \in M$ and $v \in C_m$ participating in each circle are not always the same, one node from the first such pair becomes enclosed by one of the subsequent circles.

Hence, the remaining difficulty is that we could have three such nested circles formed by nodes $m \in M$, $v \in C_m$, and three pairs of nodes from $\mathcal{N}_m(H) \cap \mathcal{N}_v(H)$ (see Figure 13.5). Any node chosen by a node $a \notin \{m, v\}$ lying on the outmost circle $X_3$ is separated from nodes enclosed by $X_1$ by $X_1$. Therefore, nodes $m' \in M$ enclosed by $X_1$ can cover only nodes that are either not adjacent to the nodes from $D_2$ considered in Lemma 13.4 (when applied to $H'$ after $X_1$ and $X_2$ already have been removed) or lie on $X_1$. Since the nodes on $X_1$ are $m$, $v$, and two of their shared neighbors in $H$, we can thus argue analogously to the proof of Lemma 13.4 in order to find a node $m'' \in M$ enclosed by $X_3$, but not enclosed by $X_1$.

Altogether, for each element of $M \cup C$ we remove at most two times two nodes each from $A$, i.e., in total no more than $4|M \cup C| \leq 28|M|$ nodes. To the remaining subgraph $H'$, we apply Lemma 13.3, yielding

$$|D_2| < (28 + 98)|M| = 126|M|. \qquad \square$$

Having determined the maximum number of nodes that enter the dominating set in each step, it remains to assemble the results and finally state the approximation ratio our algorithm achieves.

**Theorem 13.6.** $|D| < 130|M|$.

*Proof.* Combining Lemma 13.1 and Corollary 13.5, we obtain

$$|D| \leq |M| + |D_1 \setminus M| + |D_2| < (1 + 3 + 126)|M| = 130|M|. \qquad \square$$

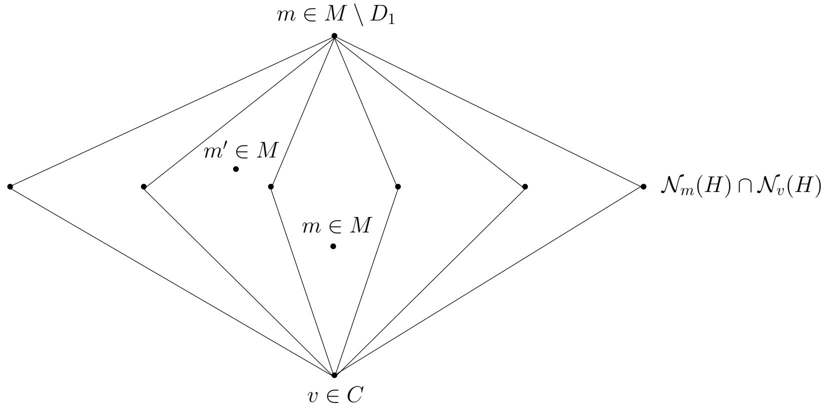

Figure 13.5: Example of a sequence of three nested circles as considered in Corollary 13.5. Each pair of two voting nodes involved in a circle is deleted from $H'$ after it has been accounted for. Therefore, all neighbors of the two outmost nodes from $\mathcal{N}_m(H) \cap \mathcal{N}_v(H)$ are not adjacent to nodes inside the innermost circle.

# Chapter 14

# Conclusions

*"I was confused and uncertain about all the little details of life. But now, while I'm still confused and uncertain, it's on a much higher plane [. . . ]" – Terry Pratchett, Equal Rites.*

In this thesis, we examined several coordination problems arising in distributed systems. We believe all presented findings to be of theoretical interest. For this reason, we would like to conclude our exposition with an educated guess on their practical significance.

In Part I, we considered the problem of synchronizing clocks in a distributed system. Chapter 4 discussed this topic in the context of wireless networks. We were able to derive and analyze a model whose implications could be validated in practice. PulseSync, the presented algorithm tailored to this model, outperforms FTSP, a commonly used synchronization protocol for sensor networks. Considering that there are ongoing efforts to form an applicable protocol out of the prototypical implementation, we hope to see first systems employing our approach in the medium term.

Less clear is the situation for our second algorithm $\mathcal{A}_\mu$ introduced in Chapter 5. However, several indicators suggest that a potential implementation could be beneficial. Firstly, the algorithm is very robust. It can tolerate worst-case crash failures, is self-stabilizing, and does not depend on a benign behavior of clock drifts or message jitters. Secondly, it has a low complexity. While protocols like PulseSync or FTSP compute a regression line out of a history of recent clock estimates, $\mathcal{A}_\mu$ follows simpler rules based on its current estimates of neighbors' clock values. These estimates can be obtained in an arbitrary manner; upper and lower bounds show that $\mathcal{A}_\mu$ makes best use of the available information up to a small factor. Thirdly, the synchronization guarantees of $\mathcal{A}_\mu$ are deterministic, which for smaller systems and/or large time frames carries the advantage of more reliable logical clocks. With these qualities, $\mathcal{A}_\mu$ might e.g. help providing distributed clock generation at hardware level (cf. [36]). Another canonical application area for $\mathcal{A}_\mu$ are highly dynamic ad-hoc networks. In such systems, the impossibility of preserving strong connectivity at all times favors an adaptive algorithm that is robust, yet ensures strong skew bounds whenever the topology is benign.

In Part II of this thesis, we analyzed the distributed balls-into-bins problem. Clearly, the practical merit of our bound of $(1 - o(1)) \log^* n$ on the time complexity of symmetric algorithms with small maximal bin load and a linear number of messages is negligible. At best, it shows that one should not invest time into the search for a constant-time solution. Similarly, the given algorithms that circumvent the lower bound are of no practical concern, as they are more complex than the simpler symmetric algorithms. In fact, the additional rounds of communication used to ensure a constant running time render these algorithms *slower* than the symmetric ones for any realistic values[1] of $n$, unless a considerable overhead in message complexity is accepted. In contrast, we assess in particular $\mathcal{A}_b^2$ notable practical relevance. In comparison to previous distributed balls-into-bins algorithms, $\mathcal{A}_b^2$ is minimalistic in terms of complexity, yet achieves a maximal bin load of two in $\log^* n + \mathcal{O}(1)$ rounds due to adaptivity. Moreover, the algorithm convinces by its robustness. Since the constant hidden in the additive term of $\mathcal{O}(1)$ is small (and can be reduced further by a more careful analysis), we estimate $\mathcal{A}_b^2$ to be well-suited for real-world use.

In Part III of our presentation, we looked into the subjects of MDS approximation and MIS computation in various graph families. For applications, it appears questionable whether the MIS algorithm from Chapter 10 is advantageous. On the one hand, it competes with the $\mathcal{O}(\log n)$ time solutions on general graphs which do not require a complicated coloring subroutine. On the other hand, our analysis yields the asymptotically stronger bound on the running time for forests only. Nevertheless, if tuned well, the algorithm might reduce running times for instances that typically occur in practice, as it might constitute an efficient heuristic also for graphs that are not forests.

Like the lower bound from Chapter 7, the $\log^*$ lower bound from Chapter 11 on the product between running time and approximation ratio of MDS approximation algorithms in unit disk graphs is a primarily theoretic statement. Similarly, the algorithm for planar graphs given in Chapter 13 is of purely theoretic interest, as it employs by far too large messages to be feasible in practice. Fascinating—and probably challenging—open questions emerging from this result are whether a constant-time $\mathcal{O}(1)$ approximation can be achieved with reasonably sized messages and what the precise approxima-

[1]For $\log^* n$ to be larger than five, $n$ must be at least $2^{65,536}$.

tion ratio is that can be achieved by distributed constant-time algorithms. Finally, from a practical viewpoint, the algorithms from Chapter 12 are more promising. Algorithm Greedy-by-Degree appeals by its low time complexity and message size. The main advantage of Algorithm Parent Dominating Set is that it achieves a constant approximation ratio on graphs of bounded arboricity. Moreover, both algorithms are considerably less complex to implement than the one from [56], making them an expedient alternative in the quite general class of graphs of small arboricity.

# Bibliography

[1] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel Randomized Load Balancing. In *Proc. 27th Symposium on Theory of Computing (STOC)*, pages 238–247, 1995.

[2] N. Alon, L. Babai, and A. Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[3] B. Awerbuch. Complexity of Network Synchronization. *Journal of the ACM*, 32(4):804–823, 1985.

[4] B. Awerbuch, Y. Azar, E. F. Grove, M.-Y. Kao, P. Krishnan, and J. S. Vitter. Load Balancing in the $L_p$ Norm. In *Proc. 36th Symposium on Foundations of Computer Science (FOCS)*, pages 383–391, 1995.

[5] B. Awerbuch and M. Sipser. Dynamic Networks are as Fast as Static Networks. In *Proc. 29th Symposium on Foundations of Computer Science (FOCS)*, pages 206–219, 1988.

[6] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced Allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[7] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the Time-Complexity of Broadcast in Multi-hop Radio Networks: An Exponential Gap Between Determinism and Randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.

[8] L. Barenboim and M. Elkin. Distributed $(\Delta+1)$-Coloring in Linear (in $\Delta$) Time. In *Proc. 41st Symposium on Theory of Computing (STOC)*, pages 111–120, 2009.

[9] L. Barenboim and M. Elkin. Sublogarithmic Distributed MIS algorithm for Sparse Graphs using Nash-Williams Decomposition. *Distributed Computing*, 22(5–6):363–379, 2009.

[10] H. Bast and T. Hagerup. Fast and Reliable Parallel Hashing. In *Proc. 3rd Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 50–61, 1991.

[11] M. Ben-Or, D. Dolev, and E. N. Hoch. Fast Self-Stabilizing Byzantine Tolerant Digital Clock Synchronization. In *Proc. 27th Symposium on Principles of Distributed Computing (PODC)*, pages 385–394, 2008.

[12] I. Ben-Zvi and Y. Moses. Beyond Lamport's *Happened-Before*: On the Role of Time Bounds in Synchronous Systems. In *Proc. 24th Symposium on Distributed Computing (DISC)*, pages 421–436, 2010.

[13] P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. W. Goldberg, Z. Hu, and R. Martin. Distributed Selfish Load Balancing. *SIAM Journal on Computing*, 37(4):1163–1181, 2007.

[14] P. Berenbrink, T. Friedetzky, Z. Hu, and R. Martin. On Weighted Balls-into-Bins Games. *Theoretical Computer Science*, 409(3):511–520, 2008.

[15] P. Berenbrink, F. Meyer auf der Heide, and K. Schröder. Allocating Weighted Jobs in Parallel. In *Proc. 9th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 302–310, 1997.

[16] S. Biaz and J. Welch. Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Information Processing Letters*, 80(3):151–157, 2001.

[17] I. Chlamtac and S. Kutten. On Broadcasting in Radio Networks – Problem Analysis and Protocol Design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.

[18] I. Chlamtac and O. Weinstein. The Wave Expansion Approach to Broadcasting in Multihop Radio Networks. *IEEE Transactions on Communications*, 39(3):426–433, 1991.

[19] R. Cole and U. Vishkin. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Information and Control*, 70(1):32–53, 1986.

[20] M. Croucher. Supercomputers Vs Mobile Phones. Walking Randomly, http://www.walkingrandomly.com/?p=2684, June 2010.

[21] A. Czumaj and W. Rytter. Broadcasting Algorithms in Radio Networks with Unknown Topology. *Journal of Algorithms*, 60(2):115–143, 2006.

[22] A. Czumaj and V. Stemann. Randomized Allocation Processes. *Random Structures and Algorithms*, 18(4):297–331, 2001.

[23] A. Czygrinow and M. Hańćkowiak. Distributed Almost Exact Approximations for Minor-Closed Families. In *Proc. 14th European Symposium on Algorithms (ESA)*, pages 244–255, 2006.

[24] A. Czygrinow and M. Hanckowiak. Distributed Approximation Algorithms for Weighted Problems in Minor-Closed Families. In *Proc. 13th Computing and Combinatorics Conference (COCOON)*, pages 515–525, 2007.

[25] A. Czygrinow, M. Hańćkowiak, and W. Wawrzyniak. Fast Distributed Approximations in Planar Graphs. In *Proc. 22nd Symposium on Distributed Computing (DISC)*, pages 78–92, 2008.

[26] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[27] D. Dolev and E. N. Hoch. Byzantine Self-Stabilizing Pulse in a Bounded-Delay Model. In *Proc. 9th Conference on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 234–252, 2007.

[28] S. Dolev and J. Welch. Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults. *Journal of the ACM*, 51:780–799, 2004.

[29] D. Dubhashi and D. Ranjan. Balls and Bins: A Study in Negative Dependence. *Random Structures and Algorithms*, 13:99–124, 1996.

[30] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization Using Reference Broadcasts. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 147–163, 2002.

[31] G. Even and M. Medina. Revisiting Randomized Parallel Load Balancing Algorithms. In *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 209–221, 2009.

[32] G. Even and M. Medina. Parallel Randomized Load Balancing: A Lower Bound for a More General Model. In *Proc. 36th Conference on Theory and Practice of Computer Science (SOFSEM)*, pages 358–369, 2010.

[33] R. Fan and N. Lynch. Gradient Clock Synchronization. In *Proc. 23rd Symposium on Principles of Distributed Computing (PODC)*, pages 320–327, 2004.

[34] F. Fich and E. Ruppert. Hundreds of Impossibility Results for Distributed Computing. *Distributed Computing*, 16(2–3):121–163, 2003.

[35] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. In *Proc. 4th Symposium on Principles of Distributed Computing (PODC)*, pages 59–70, 1985.

[36] M. Függer, U. Schmid, G. Fuchs, and G. Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proc. 6th European Dependable Computing Conference (EDCC)*, pages 87–96, 2006.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[38] J. Gil, F. Meyer auf der Heide, and A. Wigderson. The Tree Model for Hashing: Lower and Upper Bounds. *SIAM Journal on Computing*, 25(5):936–955, 1996.

[39] A. Giridhar and P. Kumar. Distributed Clock Synchronization over Wireless Networks: Algorithms and Analysis. In *Proc. 45th Conference on Decision and Control (CDC)*, pages 4915–4920, 2006.

[40] A. Goldberg, S. Plotkin, and G. Shannon. Parallel Symmetry-Breaking in Sparse Graphs. In *Proc. 19th Conference on Theory of Computing (STOC)*, pages 315–324, 1987.

[41] G. H. Gonnet. Expected Length of the Longest Probe Sequence in Hash Code Searching. *Journal of the ACM*, 28(2):289–304, 1981.

[42] T. Hagerup. The Log-Star Revolution. In *Proc. 9th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 259–278, 1992.

[43] A. Israeli and A. Itai. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22(2):77–80, 1986.

[44] R. M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM Simulation on a Distributed Memory Machine. *Algorithmica*, 16:517–542, 1996.

[45] K. Kenthapadi and R. Panigrahy. Balanced Allocation on Graphs. In *Proc. 7th Symposium on Discrete Algorithms (SODA)*, pages 434–443, 2006.

[46] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In *Proc. 6th Conference on Information Processing in Sensor Networks (IPSN)*, pages 254–263, 2007.

[47] R. Kleinberg, G. Piliouras, and E. Tardos. Load Balancing without Regret in the Bulletin Board Model. In *Proc. 28th Symposium on Principles of Distributed Computing (PODC)*, pages 56–62, 2009.

[48] E. Koutsoupias, M. Mavronicolas, and P. G. Spirakis. Approximate Equilibria and Ball Fusion. *Theory of Computing Systems*, 36(6):683–693, 2003.

[49] F. Kuhn. Weak Graph Coloring: Distributed Algorithms and Applications. In *In Proc. 21st Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 138–144, 2009.

[50] F. Kuhn, C. Lenzen, T. Locher, and R. Oshman. Optimal Gradient Clock Synchronization in Dynamic Networks. In *Proc. 29th Symposium on Principles of Distributed Computing (PODC)*, pages 430–439, 2010.

[51] F. Kuhn, C. Lenzen, T. Locher, and R. Oshman. Optimal Gradient Clock Synchronization in Dynamic Networks. *Computing Research Repository*, abs/1005.2894, 2010.

[52] F. Kuhn, T. Locher, and R. Oshman. Gradient Clock Synchronization in Dynamic Networks. In *Proc. 21st Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 270–279, 2009.

[53] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit Disk Graph Approximation. In *Proc. 2nd Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 17–23, 2004.

[54] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local Computation: Lower and Upper Bounds. *Computing Research Repository*, abs/1011.5470, 2010.

[55] F. Kuhn and R. Oshman. Gradient Clock Synchronization using Reference Broadcasts. *Computing Research Repository*, abs/0905.3454, 2009.

[56] F. Kuhn and R. Wattenhofer. Constant-Time Distributed Dominating Set Approximation. *Distributed Computing*, 17(4):303–310, 2005.

[57] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[58] C. Lenzen, T. Locher, and R. Wattenhofer. Clock Synchronization with Bounded Global and Local Skew. In *Proc. 49th Symposium on Foundations of Computer Science (FOCS)*, pages 509–518, October 2008.

[59] C. Lenzen, T. Locher, and R. Wattenhofer. Tight Bounds for Clock Synchronization. In *Proc. 28th Symposium on Principles of Distributed Computing (PODC)*, pages 46–55, August 2009.

[60] C. Lenzen, T. Locher, and R. Wattenhofer. Tight Bounds for Clock Synchronization. *Journal of the ACM*, 57(2), 2010.

[61] C. Lenzen, Y. A. Oswald, and R. Wattenhofer. What Can Be Approximated Locally? Case Study: Dominating Sets in Planar Graphs. In *Proc. 20th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 4–54, 2008.

[62] C. Lenzen, Y. A. Pignolet, and R. Wattenhofer. What Can Be Approximated Locally? Case Study: Dominating Sets in Planar Graphs. Technical report, ETH Zurich, 2010. ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-331.pdf.

[63] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal Clock Synchronization in Networks. In *Proc. 7th Conference on Embedded Networked Sensor Systems (SenSys)*, pages 225–238, 2009.

[64] C. Lenzen, J. Suomela, and R. Wattenhofer. Local Algorithms: Self-Stabilization on Speed. In *Proc. 11th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2009.

[65] C. Lenzen and R. Wattenhofer. Leveraging Linial's Locality Limit. In *Proc. 22nd Symposium on Distributed Computing (DISC)*, pages 394–407, 2008.

[66] C. Lenzen and R. Wattenhofer. Minimum Dominating Set Approximation in Graphs of Bounded Arboricity. In *Proc. 24th Symposium on Distributed Computing (DISC)*, pages 510–524, 2010.

[67] C. Lenzen and R. Wattenhofer. MIS on Trees. In *Proc. 30th Symposium on Principles of Distributed Computing (PODC)*, 2011. To appear.

[68] C. Lenzen and R. Wattenhofer. Tight Bounds for Parallel Randomized Load Balancing. *Computing Research Repository*, abs/1102.5425, 2011.

[69] C. Lenzen and R. Wattenhofer. Tight Bounds for Parallel Randomized Load Balancing. In *Proc. 43rd Symposium on Theory of Computing (STOC)*, 2011. To appear.

[70] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[71] T. Locher. *Foundations of Aggregation and Synchronization in Distributed Systems.* PhD thesis, ETH Zurich, 2009. Diss. ETH No. 18249.

[72] Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed MST for Constant Diameter Graphs. *Distributed Computing*, 18(6), 2006.

[73] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1055, 1986.

[74] J. Lundelius and N. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2/3):190–204, 1984.

[75] N. Lynch. A Hundred Impossibility Proofs for Distributed Computing. In *Proc. 8th Symposium on Principles of Distributed Computing (PODC)*, pages 1–28, 1989.

[76] M. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple Heuristics for Unit Disk Graphs. *Journal of Networks*, 25:59–68, 1995.

[77] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *Proc. 2nd Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–49, 2004.

[78] Y. Matias and U. Vishkin. Converting High Probability into Nearly-Constant Time—with Applications to Parallel Hashing. In *Proc. 23rd Symposium on Theory of Computing (STOC)*, pages 307–316, 1991.

[79] L. Meier and L. Thiele. Gradient Clock Synchronization in Sensor Networks. Technical report, ETH Zurich, 2005. TIK Report 219.

[80] Y. Métivier, J. M. Robson, N. Saheb Djahromi, and A. Zemmari. An optimal bit complexity randomised distributed MIS algorithm. In *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 323–337, 2009.

[81] F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting Storage Redundancy to Speed up Randomized Shared Memory Simulations. *Theoretical Computer Science*, 162(2):245–281, 1996.

[82] M. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing.* PhD thesis, University of California, Berkeley, 1996.

[83] M. Mitzenmacher. How Useful is Old Information? Technical report, Systems Research Center, Digital Equipment Corporation, 1998.

[84] M. Mitzenmacher. On the Analysis of Randomized Load Balancing Schemes. In *Proc. 10th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 292–301, 1998.

[85] M. Mitzenmacher, B. Prabhakar, and D. Shah. Load Balancing with Memory. In *Proc. 43th Symposium on Foundations of Computer Science (FOCS)*, pages 799–808, 2002.

[86] M. Mitzenmacher, A. Richa, and R. Sitaraman. *Handbook of Randomized Computing*, volume 1, chapter The Power of Two Random Choices: A Survey of the Techniques and Results, pages 255–312. Kluwer Academic Publishers, Dordrecht, 2001.

[87] T. Moscibroda, P. von Rickenbach, and R. Wattenhofer. Analyzing the Energy-Latency Trade-Off During the Deployment of Sensor Networks. In *Proc. 25th Conference on Computer Communications (INFOCOM)*, pages 2492–2504, 2006.

[88] T. Moscibroda and R. Wattenhofer. The Complexity of Connectivity in Wireless Networks. In *Proc. 25th Conference on Computer Communications (InfoCom)*, pages 25–37, 2006.

[89] M. Naor. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991.

[90] A. Panconesi and A. Srinivasan. On the Complexity of Distributed Network Decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

[91] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

[92] D. Peleg and V. Rubinovich. A Near-Tight Lower Bound on the Time Complexity of Distributed MST Construction. In *Proc. 40th Symposium on Foundations of Computer Science (FOCS)*, pages 253–261, 1999.

[93] Y. Peres, K. Talwar, and U. Wieder. The $(1+\beta)$-Choice Process and Weighted Balls-into-Bins. In *Proc. 21th Symposium on Discrete Algorithms (SODA)*, pages 1613–1619, 2010.

[94] R. Raman. The Power of Collision: Randomized Parallel Algorithms for Chaining and Integer Sorting. In *Proc. 10th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 161–175, 1990.

[95] R. Raz and S. Safra. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP. In *Proc. 29th Symposium on Theory of Computing (STOC)*, pages 475–484, 1997.

[96] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. B. Srivastava, and P. Dutta. A Case Against Routing-Integrated Time Synchronization. In *Proc. 8th Conference on Embedded Networked Sensor Systems (SenSys)*, pages 267–280, 2010.

[97] T. Schmid, P. Dutta, and M. B. Srivastava. High-Resolution, Low-Power Time Synchronization an Oxymoron no More. In *Proc. 9th Conference on Information Processing in Sensor Networks (IPSN)*, pages 151–161, 2010.

[98] J. Schneider and R. Wattenhofer. A log-star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Proc. 27th Symposium on Principles of Distributed Computing (PODC)*, pages 35–44, 2008.

[99] T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, 1987.

[100] V. Stemann. Parallel Balanced Allocations. In *Proc. 8th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 261–269, 1996.

[101] A. A. Syed and J. Heidemann. Time Synchronization for High Latency Acoustic Networks. In *Proc. 25th Conference on Computer Communications (InfoCom)*, pages 892–903, 2006.

[102] K. Talwar and U. Wieder. Balanced Allocations: The Weighted Case. In *Proc. 39th Symposium on Theory of Computing (STOC)*, pages 256–265, 2007.

[103] B. Vöcking. How Asymmetry Helps Load Balancing. *Journal of the ACM*, 50(4):568–589, 2003.

[104] U. Wieder. Balanced Allocations with Heterogenous Bins. In *Proc. 19th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 188–193, 2007.

# Curriculum Vitae

| | |
|---|---|
| June 12, 1982 | Born in Kleve, Germany |
| 1988–2001 | Primary and secondary education in Goch, Germany |
| 2001–2002 | Civilian service, Goch, Germany |
| 2002–2007 | Studies in mathematics, University of Bonn, Germany |
| October 2007 | Diploma in mathematics, University of Bonn, Germany |
| 2007–2010 | Ph.D. student, research and teaching assistant, Distributed Computing Group, Prof. Roger Wattenhofer, ETH Zurich, Switzerland |
| January 2011 | Ph.D. degree, Distributed Computing Group, ETH Zurich, Switzerland<br>Advisor: Professor Roger Wattenhofer<br>Co-examiners: Professor Danny Dolev, University of Jerusalem, Israel<br>Professor Berthold Vöcking, University of Aachen, Germany |

# Publications

The following list comprises the publications[2] I co-authored during my time with the Distributed Computing Group at ETH Zurich.

1. **Tight Bounds for Parallel Randomized Load Balancing.** Christoph Lenzen and Roger Wattenhofer. *43rd Symposium on Theory of Computing (STOC).* San Jose, California, USA, June 2011.

2. **MIS on Trees.** Christoph Lenzen and Roger Wattenhofer. *30th Symposium on Principles of Distributed Computing (PODC).* San Jose, California, USA, June 2011.

3. **Minimum Dominating Set Approximation in Graphs of Bounded Arboricity.** Christoph Lenzen and Roger Wattenhofer. *24th Symposium on Distributed Computing (DISC).* Cambridge, Massachusetts, USA, September 2010.

4. **Optimal Gradient Clock Synchronization in Dynamic Networks.** Fabian Kuhn, Christoph Lenzen, Thomas Locher, and Rotem Oshman. *29th Symposium on Principles of Distributed Computing (PODC).* Zurich, Switzerland, July 2010.

5. **Brief Announcement: Exponential Speed-Up of Local Algorithms using Non-Local Communication.** Christoph Lenzen and Roger Wattenhofer. *29th Symposium on Principles of Distributed Computing (PODC).* Zurich, Switzerland, July 2010.

6. **Tight Bounds for Clock Synchronization**. Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. *Journal of the ACM, 57(2).* January 2010.

[2]Technical reports accompanying conference papers are not listed separately.

7. **Clock Synchronization: Open Problems in Theory and Practice.** Christoph Lenzen, Thomas Locher, Philipp Sommer, and Roger Wattenhofer. *36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM).* Špindlerův Mlýn, Czech Republic, January 2010.

8. **A Review of PODC 2009.** Keren Censor and Christoph Lenzen. *Distributed Computing Column 36, SIGACT News 40(4).* December 2009.

9. **Local Algorithms: Self-Stabilization on Speed.** Christoph Lenzen, Jukka Suomela, and Roger Wattenhofer. *11th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS).* Lyon, France, November 2009.

10. **Optimal Clock Synchronization in Networks.** Christoph Lenzen, Philipp Sommer, and Roger Wattenhofer. *7th Conference on Embedded Networked Sensor Systems (SenSys).* Berkeley, California, USA, November 2009.

11. **Tight Bounds for Clock Synchronization.** Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. *28th Symposium on Principles of Distributed Computing (PODC).* Calgary, Canada, August 2009.

12. **Clock Synchronization with Bounded Global and Local Skew.** Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. *49th Symposium on Foundations of Computer Science (FOCS).* Philadelphia, Pennsylvania, USA, October 2008.

13. **Leveraging Linial's Locality Limit.** Christoph Lenzen and Roger Wattenhofer. *22nd Symposium on Distributed Computing (DISC).* Arcachon, France, September 2008.

14. **What Can Be Approximated Locally? Case Study: Dominating Sets in Planar Graphs.** Christoph Lenzen, Yvonne Anne Oswald, and Roger Wattenhofer. *20th Symposium on Parallelism in Algorithms and Architectures (SPAA).* Munich, Germany, June 2008.