# Metastability-Containing Circuits, Parallel Distance Problems, and Terrain Guarding

A dissertation submitted towards the degree Doctor of Engineering of the Faculty of Mathematics and Computer Science of Saarland University

by Stephan Friedrichs

Saarbrücken / 2017

| | |
|---|---|
| Day of Colloquium: | 11. September 2017 |
| Dean of the Faculty: | Univ.-Prof. Dr. Frank-Olaf Schreyer |
| | |
| Chair of the Committee: | Prof. Dr.-Ing. Holger Hermanns |
| Reporters | |
| First reviewer: | Dr. Christoph Lenzen |
| Second reviewer: | Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn |
| Third reviewer: | Prof. Dr. Mohsen Ghaffari |
| Academic Assistant: | Dr. Emanuele Natale |

# Abstract

*"Don't quote me on that!"*

Christoph Lenzen

**Abstract** We study three problems. The first is the phenomenon of metastability in digital circuits. This is a state of bistable storage elements, such as registers, that is neither logical 0 nor 1 and breaks the abstraction of Boolean logic. We propose a time- and value-discrete model for metastability in digital circuits and show that it reflects relevant physical properties. Further, we propose the fundamentally new approach of using logical masking to perform meaningful computations despite the presence of metastable upsets and analyze what functions can be computed in our model. Additionally, we show that circuits with masking registers grow computationally more powerful with each available clock cycle.

The second topic are parallel algorithms, based on an algebraic abstraction of the Moore-Bellman-Ford algorithm, for solving various distance problems. Our focus are distance approximations that obey the triangle inequality while at the same time achieving polylogarithmic depth and low work.

Finally, we study the continuous Terrain Guarding Problem. We show that it has a rational discretization with a quadratic number of guard candidates, establish its membership in NP and the existence of a PTAS, and present an efficient implementation of a solver.

**Zusammenfassung** Wir betrachten drei Probleme, zunächst das Phänomen von Metastabilität in digitalen Schaltungen. Dabei geht es um einen Zustand in bistabilen Speicherelementen, z. B. Registern, welcher weder logisch 0 noch 1 entspricht und die Abstraktion Boolescher Logik unterwandert. Wir präsentieren ein zeit- und wertdiskretes Modell für Metastabilität in digitalen Schaltungen und zeigen, dass es relevante physikalische Eigenschaften abbildet. Des Weiteren präsentieren wir den grundlegend neuen Ansatz, trotz auftretender Metastabilität mit Hilfe von logischem Maskieren sinnvolle Berechnungen durchzuführen und bestimmen, welche Funktionen in unserem Modell berechenbar sind. Darüber hinaus zeigen wir, dass durch Maskingregister in zusätzlichen Taktzyklen mehr Funktionen berechenbar werden.

Das zweite Thema sind parallele Algorithmen die, basierend auf einer Algebraisierung des Moore-Bellman-Ford-Algorithmus, diverse Distanzprobleme lösen. Der Fokus liegt auf Distanzapproximationen unter Einhaltung der Dreiecksungleichung bei polylogarithmischer Tiefe und niedriger Arbeit.

Abschließend betrachten wir das kontinuierliche Terrain Guarding Problem. Wir zeigen, dass es eine rationale Diskretisierung mit einer quadratischen Anzahl von Wächterpositionen erlaubt, folgern dass es in NP liegt und ein PTAS existiert und präsentieren eine effiziente Implementierung, die es löst.

# Acknowledgments

*"Moin Jungs, zwei Caipis?"*

The Feinkost barkeeper

I want to express my sincere gratitude to my advisor *Christoph Lenzen.* Thank you for advising with ample patience, knowing when to ask new research questions and when to let me to pursue my own projects, and for making the Theory of Distributed and Embedded Systems group a wonderful place to work at, creating an atmosphere of research, learning, writing, teaching, traveling, and various social activities. I'm indebted to *Kurt Mehlhorn* for offering me a position at the Max Planck Institute for Informatics' Algorithms and Complexity group, it has been an incredibly rewarding time. I want to thank *Mohsen Ghaffari* for taking the time to be the external reviewer.

Out of my colleagues and friends at the MPI, I want to mention some in particular. *Joel Rybicki,* thanks for exploring the city with me, helping me figure out the tabs of Any Other Day, and countless hot and cold beverages. Thank you *Matthias Függer* for great research, living the culture of the research coffee, the afternoons with guitars in the park, and numerous visits to Feinkost, Nautilus, Vienna and Paris. I wish to thank *Attila Kinali* for establishing the chocolate supply-chain, being a great office mate, lively discussions, and patiently explaining the dos and don'ts of digital circuit design to me. *Moti Medina,* thank you for great research, drinks and movies, and always knowing which is advisable.

Regarding my time in Braunschweig, I want to thank my friend *Christiane Schmidt* for always having an open ear, co-founding the research Friday, and all the support. Thank you *Michael Hemmer,* co-founder of the research Friday, for always having my back and being a great host, colleague, and friend. I also wish to thank my friend *Henning Hasemann* for lots of hot and cold beverages and the music. To *Alexander Kröller* I owe thanks for the support and for convincing me to take part in the teach4tu program.

I'm indebted to *Christoph Lenzen, Matthias Függer, Attila Kinali, Christiane Schmidt, Michael Hemmer, Christian Ikenmeyer,* and *Pavel Kolev* for many helpful comments on this thesis.

As important as a healthy professional environment is, it cannot substitute a stable personal environment, friends with open ears and good advice, and a supporting family. Hence, I want to thank my friends, *Henning Günther, Martin Wegner, Jaffi Schrader* and *Thomas Hoffmann,* and my family, *Frauke, Werner, Karsten* and *Bernhard.*

# Contents

*"The most merciful thing in the world, I think, is the inability of the human mind to correlate all its contents."*

H. P. Lovecraft, The Call of Cthulhu

# CHAPTER 1
# Introduction

*"The boundaries between true intellectual disciplines are currently enforced by little more than university budgets and architecture."*

Sam Harris, Our Narrow Definition of "Science"

While this thesis is firmly rooted in theoretical computer science, we study problems from three vastly different areas — digital circuit design, parallel algorithms, and computational geometry — using very different techniques and asking very different questions. Our study of digital circuit design is motivated from electrical engineering and requires us to look at transistor-level circuits. At the same time, we ask questions from the realm of theoretical computer science: What can be computed in our circuit model w.r.t. a range of parameters? Regarding parallel algorithms, we design algorithms for distance problems in graphs in a theoretical model of computation. Our approach, however, blends into distributed computing and employs algebraic techniques. In the third part, we study a popular problem from computational geometry and answer important theoretical questions about it. Our results make an implementation possible. We develop an implementation and test it with exhaustive computational experiments. Observations from such experiments feed back into the theoretical contributions, refining them in service of an optimized implementation.

This thesis is organized in three parts. The parts are self-contained, except for notation and preliminaries presented in Chapter 2 and the assumption that the reader is familiar with the foundations of theoretical computer science. Below, we informally motivate each part; an in-depth introduction and discussion of related work is presented in the respective introductory chapters.

**Metastability-Containing Circuits** Digital circuits are an integral part of virtually every contemporary piece of technology. They are used in, e.g., desktop computers, servers, phones, and network routers, as well as in machines under some degree of electronic control, like cars, trains, planes, washing machines, medical equipment, robots, and coffee machines.

At its heart, a digital circuit relies on a simple concept: Two distinct voltage levels represent logical 0 and logical 1. Unfortunately, intermediate voltages cannot be avoided entirely: The output voltage behaves continuously over time, hence, sampling at a critical moment results in a deteriorated signal. This is unavoidable in, e.g., analog-to-digital conversions or when communicating across unsynchronized clock domains.

Ultimately, these effects lead to deteriorated signals, unspecified voltage levels, and degraded timing behavior. These, in turn, can drive memory elements like registers into metastability, a state of a bistable element. Metastable registers typically output unspecified voltages between logical 0 and 1 or show late transitions, i.e., output further deteriorated signals. In other words, metastability causes deterioration, which causes metastability, etc. This can continue indefinitely and render the abstraction of Boolean logic meaningless.

The standard approach to solve this issue is to exploit that the probability of maintained metastability decreases exponentially over time, i.e., to wait. This is implemented with specialized storage elements called synchronizers. Unfortunately, this merely decreases the odds of metastability instead of eliminating it altogether and delays the computation. Furthermore, trends in hardware design like growth in circuit complexity, reduced operating voltage, and increasing numbers of clock domains intensify the risk of metastable upsets.

In Part I, we study a novel perspective on the phenomenon of metastability. Our key technique is logical masking: While the output of an AND gate with inputs M and 1 — where M represents a deteriorated signal — is $\mathrm{M} \wedge 1 = \mathrm{M}$, its output under inputs M and 0 is $\mathrm{M} \wedge 0 = 0$. In the latter case, the stable input of 0 fixes the output and voids the deteriorated input's influence on the output. This is equivalent to Kleene's three-valued logic. Instead of trying to probabilistically shield a circuit against metastability, we accept that metastability cannot be prevented and contain its effects using logical masking.

Our core contribution is a clocked time- and value-discrete model for metastability in digital circuits. We verify that it reflects known physical phenomena, i.e., the impossibility of avoiding, detecting, or resolving metastability in a digital circuit, and show that non-trivial positive results can be derived in it. As our model assumes worst-case propagation of metastability, we can derive deterministic guarantees as opposed to the probabilistic promises provided by synchronizers.

On the practical side, we establish that a variety of metastability-containing components like Metastability-Containing Multiplexers (CMUXes), unary to Gray code converters, and sorting networks can be implemented. Regarding CMUXes, we present efficient transistor-level implementations. The list of presented components is by no means exhaustive, but already establishes that a synchronizer-free implementation of a fault-tolerant clock-synchronization algorithm is within reach. Furthermore, this establishes that synchronization delay poses no fundamental limit on the operating frequency of clock-synchronization hardware.

On the theoretical side, we examine which functions can be implemented in our circuit model. Our key result in this regard is that the number of clock cycles is irrelevant in combinational circuits and in circuits restricted to standard registers, but that the class of computable functions strictly grows with each clock cycle when permitting masking registers. Furthermore, we classify all functions computable by combinational circuits and circuits without masking registers.

**Parallel Distance Problems** Distance problems in graphs are ubiquitous in computer science. This is at least partly due to the fact that graphs are such a general tool; applications include navigation, logistics, digital circuit design, networks of neurons, social networks, and many more. Another core problem in computer science, the relevance of which increased with the availability of multi-core CPUs, is parallelization. Hence, we study parallel distance problems in Part II.

To this end, we turn our attention to the classical Moore-Bellman-Ford (MBF) algorithm, which offers simplicity, generalizes to a variety of distance problems, and is benign in terms of parallelization. We propose an algebraic generalization of the MBF algorithm and demonstrate that it successfully describes a wide range of known algorithms, which we term Moore-Bellman-Ford-like (MBF-like) algorithms. What MBF-like algorithms have in common is their strategy to spread information through a network. Besides being of theoretical interest and offering a new perspective on the MBF algorithm, MBF-like algorithms prove useful for making statements of the form "$d$ appropriately arranged iterations in graph $G$ are equivalent to an iteration in graph $H$" tangible while at the same time separating them from concrete algorithms.

We develop parallel algorithms of polylogarithmic depth (parallel time) and little work (sequential time) that determine approximate metrics or metric tree embeddings of weighted graphs. As these are fundamental tools for approximating distance problems, it is important to develop efficient parallel algorithms for them.

Our core algorithmic result is an oracle for MBF-like queries. Provided with an MBF-like algorithm and a graph $G$, the oracle answers with the result of the algorithm in a graph $H$ which approximates distances of $G$. While $H$ has a low Shortest-Path Diameter (SPD), which is key to polylogarithmic-depth algorithms, we cannot run the algorithm in $H$ directly: $H$ is a complete graph and hence explicitly constructing it is computationally too expensive. The insight is that the oracle can simulate iterations in $H$ using only iterations in $G$ and a polylogarithmic overhead in both depth and work.

As a consequence of our techniques, we are able to efficiently construct approximate metrics and sample tree embeddings of expected logarithmic stretch by querying the oracle with appropriate MBF-like algorithms. Furthermore, our techniques allow us to improve upon the state of the art regarding the distributed computation of metric tree embeddings as well as the parallel approximation of the buy-at-bulk network design and $k$-median problems.

**Terrain Guarding** Many real-world problems are intrinsically linked to geometry, especially when the entities associated with them have a location, shape, or orientation in space. Examples for this include the optimization of a parking-lot layout where access routes must be kept free, various problems in robotics, and place & route in circuit design.

A famous example is the Art Gallery Problem (AGP), where one is given a 2D floor plan of a building, the art gallery, and looks for a minimum number of security cameras that together cover the entire gallery. In its classical 2D form, the AGP lacks height information. Adding such information, creating a 2.5D AGP, is highly relevant — e.g., to optimally cover an outdoor environment that includes hills, trees, and buildings with cellphone towers — but at the same time a non-trivial endeavor.

Part III discusses the Terrain Guarding Problem (TGP). It is a relative of the AGP that is all about height information and may hence serve as an intermediate step towards a 2.5D AGP. In the TGP, we remove one dimension from the AGP and instead add height. The result is an $x$-monotone chain of line segments, like the altitude profile of a mountain road, which is to be guarded.

Our main result is that the continuous version of the TGP, where there is no restriction on where on the terrain the guards may be placed, has a discretization of $O(n^2)$ potential guard locations, where $n$ is the number of vertices of the terrain. The coordinates of our discretization are rational if the terrain's vertex coordinates are. This is surprising, because the AGP does not permit such a result, not even for monotone polygons. We hope that our insights serve as a stepping stone towards a 2.5D AGP. Due to previous work, the existence of such a discretization implies that the continuous TGP admits a Polynomial-Time Approximation Scheme (PTAS) and is NP-complete.

In addition to theoretical insights, we present an algorithm that efficiently solves large instances of the TGP. To this end, we combine theory and practice, devise an efficient C++ implementation, make effective use of the Computational Geometry Algorithms Library (CGAL), apply linear optimization techniques, significantly reduce the size of our discretization, and thoroughly evaluate our implementation in experiments.

# CHAPTER 2

# Notation and Preliminaries

*"The scientific theory I like best is that the rings of Saturn are composed entirely of lost airline luggage."*

Mark Russell

In order to avoid ambiguities, we denote the natural numbers with and without 0 by $\mathbb{N}_0$ and $\mathbb{N}$, respectively. The integers are denoted by $\mathbb{Z}$, the rationals by $\mathbb{Q}$, and the real numbers by $\mathbb{R}$. For $c \in \mathbb{R}$, we occasionally use $\mathbb{R}_{\geq c}$ to refer to the set $\{x \in \mathbb{R} \mid x \geq c\}$; $\mathbb{R}_{>c}$, $\mathbb{R}_{\leq c}$, and $\mathbb{R}_{<c}$ are defined analogously. Furthermore — usually in the context of Boolean expressions — we use $\mathbb{B} := \{0, 1\}$, where we associate 0 with *false* and 1 with *true,* if applicable. At times, we use the shorthand $\bar{x} := 1 - x$ for $x \in \mathbb{B}$.

Let $M$ and $N$ be sets. For $c \in N$, $\operatorname{const}_c\colon M \to N$ is the constant function with $\operatorname{const}_c(x) = c$. We refer to $\operatorname{id}\colon M \to N$ as the identity function with $\operatorname{id}(x) = x$, which requires $M \subseteq N$. Furthermore, we denote by $\mathcal{P}(M) := \{M' \subseteq M\}$ the power set of $M$, and by $\binom{N}{k} := \{N' \subseteq N \mid |N'| = k\}$ all subsets of $N$ with cardinality $k \in \mathbb{N}_0$.

For a totally ordered non-empty set $M$, we use $\min, \max\colon \mathcal{P}(M) \setminus \emptyset \to M$ as functions that pick the minimum and maximum of a set, respectively. Whenever convenient, however, we use them as binary operators $\min, \max\colon M \times M \to M$, e.g., as the operation of a semigroup.

We frequently use asymptotic notation to reason about various quantities. Consider a function $f\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ with $n \mapsto f(n)$. We use the standard definitions of $\mathrm{O}(f)$, $\Omega(f)$, $\Theta(f)$, $\mathrm{o}(f)$, and $\omega(f)$ provided by, e.g., Cormen et al. [35]. Furthermore, we abbreviate the set of polynomials in $n$ by $\operatorname{poly} n$ and the set of polylogarithms in $n$ by $\operatorname{polylog} n$. In the context of problem size $n$, we use $\tilde{\mathrm{O}}(f)$, $\tilde{\Omega}(f)$, $\tilde{\Theta}(f)$, $\tilde{\mathrm{o}}(f)$, and $\tilde{\omega}(f)$ to hide polylogarithmic factors in $n$; an example is $\tilde{\mathrm{O}}(f) := \mathrm{O}(f) \cdot \operatorname{polylog} n$, the other sets are defined analogously.

## 2.1 Algebraic Foundations

For the sake of self-containment and unambiguousness, we give some algebraic definitions and a standard result. Definitions 2.1–2.3 are adapted from Chapters 1 and 5 of Hebisch and Weinert [72].

Semigroups are among the most fundamental structures; they merely comprise a set $M$ of elements and an associative operation $\circ\colon M \times M \to M$. Commutativity and the existence of a neutral element can optionally be required.

**Definition 2.1** (Semigroup). *Let $M$ be a set and $\circ\colon M \times M \to M$ a binary operation. $(M, \circ)$ is a* semigroup *if and only if $\circ$ is associative, i.e.,*

$$\forall x, y, z \in M\colon \quad x \circ (y \circ z) = (x \circ y) \circ z. \tag{2.1}$$

*We write $x \in (M, \circ)$ for $x \in M$. A semigroup $(M, \circ)$ is* commutative *if and only if*

$$\forall x, y \in M\colon \quad x \circ y = y \circ x. \tag{2.2}$$

*$e \in M$ is a* neutral element *of $(M, \circ)$ if and only if*

$$\forall x \in M\colon \quad e \circ x = x \circ e = x. \tag{2.3}$$

Semirings — rings without additive inverses — are structures that tie together two semigroups on the same set of elements. Hence, there are two operations, referred to as *addition* and *multiplication.* These operations are required to obey the distributive laws we are accustomed to from, e.g., $(\mathbb{N}_0, +, \cdot)$, $(\mathbb{Z}, +, \cdot)$, $(\mathbb{Q}, +, \cdot)$, and $(\mathbb{R}, +, \cdot)$. Further examples for semirings are $\mathcal{S}_{\min,+} = (\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +)$, known as the min-plus or the tropical semiring, and the Boolean semiring $(\mathbb{B}, \vee, \wedge)$. $(\mathbb{N}, +, \cdot)$ is not a semiring due to our convention of $0 \notin \mathbb{N}$, i.e., because it lacks the neutral element of addition.

In the context of addition and multiplication, we refer to neutral elements as 0 and 1, respectively. Note, however, that the meaning of 0 and 1 depends on the concrete operations. Consider $\mathcal{S}_{\min,+}$ as an example: Its zero, the neutral element of addition (min), is $\infty$ and its one, the neutral element of multiplication $(+)$, is 0.

**Definition 2.2** (Semiring). *Let $S \neq \emptyset$ be a set, and $\oplus, \odot\colon S \times S \to S$ binary operations. Then $\mathcal{S} = (S, \oplus, \odot)$ is a* semiring *if and only if*

*(1) $(S, \oplus)$ is a commutative semigroup with neutral element* 0*,*

*(2) $(S, \odot)$ is a semigroup with neutral element* 1*,*

*(3) the left- and right-distributive laws hold:*

$$\forall x, y, z \in S\colon \quad x \odot (y \oplus z) = (x \odot y) \oplus (x \odot z) \text{ and} \tag{2.4}$$
$$\forall x, y, z \in S\colon \quad (y \oplus z) \odot x = (y \odot x) \oplus (z \odot x), \text{ and} \tag{2.5}$$

*(4)* 0 annihilates *w.r.t. $\odot$:*

$$\forall x \in S\colon \quad 0 \odot x = x \odot 0 = 0. \tag{2.6}$$

*We write $x \in \mathcal{S}$ for $x \in S$, and refer to $\oplus$ as* addition *and to $\odot$ as* multiplication.

Some authors do not require semirings to have neutral elements or an annihilating 0. We, however, require them by definition because we need them and work on semirings which provide them.

Semimodules over semirings generalize vector spaces over fields. The elements of a semimodule have an addition and support scalar multiplication with the semiring elements; both operations obey distributive laws.

**Definition 2.3** (Semimodule)**.** *Let $\mathcal{S} = (S, \oplus, \odot)$ be a semiring. $\mathcal{M} = (M, \oplus, \odot)$ with binary operations $\oplus\colon M \times M \to M$ and $\odot\colon S \times M \to M$ is a* semimodule over $\mathcal{S}$ *if and only if*

*(1) $(M, \oplus)$ is a semigroup and*

*(2) for all $s, t \in S$ and all $x, y \in M$:*

$$1 \odot x = x, \tag{2.7}$$
$$s \odot (x \oplus y) = (s \odot x) \oplus (s \odot y), \tag{2.8}$$
$$(s \oplus t) \odot x = (s \odot x) \oplus (t \odot x), \text{ and} \tag{2.9}$$
$$(s \odot t) \odot x = s \odot (t \odot x). \tag{2.10}$$

*$\mathcal{M}$ is* zero-preserving *if and only if*

*(1) $(M, \oplus)$ has the neutral element $0$ and*

*(2) $0 \in S$ is an* annihilator *for $\odot$:*

$$\forall x \in M\colon \quad 0 \odot x = 0. \tag{2.11}$$

*We write $x \in \mathcal{M}$ for $x \in M$, and refer to $\oplus$ as* addition *and to $\odot$ as* (scalar) multiplication.

In the context of a semiring or semigroup, we implicitly associate $\oplus$ and $\odot$ with the respective addition and (scalar) multiplication. Furthermore, we follow the convention to occasionally omit $\odot$ and give it priority over $\oplus$, for example, we interpret $xy \oplus z$ as $(x \odot y) \oplus z$.

A common semimodule over the semiring $\mathcal{S}$ is $\mathcal{S}^k$ with coordinatewise addition, i.e., $k$-dimensional vectors over $\mathcal{S}$. Note that $\mathcal{S} = \mathcal{S}^1$ always is a semimodule over itself. The following lemma is a standard result.

**Lemma 2.4.** *Let $\mathcal{S} = (S, \oplus, \odot)$ be a semiring and $k \in \mathbb{N}_0$ an integer. Then $\mathcal{S}^k := (S^k, \oplus, \odot)$ with, for all $s \in S$ and $x, y \in S^k$,*

$$(x \oplus y)_i := x_i \oplus y_i \text{ and} \tag{2.12}$$
$$(s \odot x)_i := s \odot x_i \tag{2.13}$$

*is a zero-preserving semimodule over $\mathcal{S}$ with zero $(0, \dots, 0)^\top$.*

*Proof.* We check the conditions of Definition 2.3 one by one. Throughout the proof, let $1 \le i \le k$, $s, t \in S$, and $x, y \in S^k$ be arbitrary.

(1) $(S^k, \oplus)$ is a semigroup because $(S, \oplus)$ is.

(2) Equations (2.7)–(2.10) hold due to

$$(1 \odot x)_i = 1 \odot x_i = x_i, \tag{2.14}$$
$$(s \odot (x \oplus y))_i = s \odot (x_i \oplus y_i) = (s \odot x_i) \oplus (s \odot y_i) = ((s \odot x) \oplus (s \odot y))_i, \tag{2.15}$$
$$((s \oplus t) \odot x)_i = (s \oplus t) \odot x_i = (s \odot x_i) \oplus (t \odot x_i) = ((s \odot x) \oplus (t \odot x))_i, \tag{2.16}$$
$$\text{and } ((s \odot t) \odot x)_i = (s \odot t) \odot x_i = s \odot (t \odot x_i) = (s \odot (t \odot x))_i. \tag{2.17}$$

(3) $(0, \dots, 0)$ is the neutral element of $(S^k, \oplus)$ because 0 is the neutral element of $(S, \oplus)$.

(4) 0 is an annihilator for $\odot$:

$$(0 \odot x)_i = 0 \odot x_i = 0. \tag{2.18}$$

□

## 2.2 Probability Theory

We refer to standard literature [35, 114] for the basics of probability theory. The following is adapted from Mitzenmacher and Upfal [114]. Given a *sample space* $\Omega$ and an *event* $\mathcal{E} \subseteq \Omega$, we write $\mathbb{P}[\mathcal{E}]$ for the *probability* of $\mathcal{E}$. The *complement* of $\mathcal{E}$ is $\bar{\mathcal{E}} := \Omega \setminus \mathcal{E}$. Regarding a *random variable* $X\colon \Omega \to \mathbb{R}$, we denote its *expectation* by $\mathbb{E}[X]$.

A first simple, useful, and ubiquitous bound is the union bound. We adapt the version of Mitzenmacher and Upfal [114].

**Lemma 2.5** (Union Bound)**.** *Let $\mathcal{E}_1, \mathcal{E}_2, \dots$ be countably many events. Then we have*

$$\mathbb{P}\left[\bigcup_{i \geq 1} \mathcal{E}_i\right] \leq \sum_{i \geq 1} \mathbb{P}\left[\mathcal{E}_i\right]. \tag{2.19}$$

The next concept we need is a standard definition of events occurring *with high probability.*

**Definition 2.6** (With High Probability)**.** *Let $\mathcal{E}$ be an event. $\mathcal{E}$ occurs* with high probability (w.h.p.) *if $\mathbb{P}[\mathcal{E}] \geq 1 - n^{-c}$ for any constant $c \in \mathbb{R}_{\geq 1}$ that is fixed in advance.*

Observe that $c$ is a constant in terms of the O-notation. The idea is to design an algorithm $\mathcal{A}$ that is parameterized with $c$ and succeeds w.h.p., which is much stronger than an algorithm $\mathcal{A}'$ that succeeds with a constant probability $p < 1$. As an example, suppose that $\mathcal{A}'$ is used as a subroutine that is invoked $k$ times and has to succeed every time. The success probability of that quickly becomes small when $k$ grows. Using $\mathcal{A}$ instead, $c$ can be chosen such that all $k$ instances of $\mathcal{A}$ collectively succeed w.h.p. as long as $k \in \operatorname{poly} n$, which is a standard result:

**Lemma 2.7.** *Let $\mathcal{E}_1, \dots, \mathcal{E}_k$ be events occurring w.h.p. and $k \in \operatorname{poly} n$. Then the event $\mathcal{E}_1 \cap \dots \cap \mathcal{E}_k$ occurs w.h.p.*

*Proof.* If $n = 1$ the claim is trivial. Otherwise, we have $k \leq an^b$ for fixed $a, b \in \mathbb{R}_{>0}$ and choose that all $\mathcal{E}_i$ occur with a probability of at least $1 - n^{-c'}$ with $c' := c + b + \log_2 a$ for some fixed $c \in \mathbb{R}_{\geq 1}$. As $\log_2 a \geq \log_n a$, the union bound yields

$$\mathbb{P}\left[\overline{\mathcal{E}_1 \cap \dots \cap \mathcal{E}_k}\right] \overset{(2.19)}{\leq} \sum_{i=1}^{k} \mathbb{P}\left[\bar{\mathcal{E}}_i\right] \leq kn^{-c'} \leq an^b n^{-c-b-\log_n a} = n^{-c}, \tag{2.20}$$

hence $\mathcal{E}_1 \cap \dots \cap \mathcal{E}_k$ occurs w.h.p. as claimed. □

Chernoff's bound is a commonly used, strong bound regarding the sum of independent 0–1 random variables. We adapt the version of Mitzenmacher and Upfal [114].

**Lemma 2.8** (Chernoff's Bound)**.** *Let $X_1, \dots, X_n$ be independent* 0–1 *random variables and $X := \sum_{i=1}^{n} X_i$. Then the following bounds hold.*

*(1) For all $\delta \in \mathbb{R}_{>0}$*

$$\mathbb{P}\left[X \geq (1+\delta)\,\mathbb{E}[X]\right] \quad \leq \quad \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mathbb{E}[X]}, \tag{2.21}$$

*(2) for all $0 < \delta \leq 1$*

$$\mathbb{P}\left[X \geq (1+\delta)\,\mathbb{E}[X]\right] \quad \leq \quad e^{-\,\mathbb{E}[X]\delta^2/3},\ \textit{and} \tag{2.22}$$

*(3) for all $R \geq 6\,\mathbb{E}[X]$*

$$\mathbb{P}\left[X \geq R\right] \quad \leq \quad 2^{-R}. \tag{2.23}$$

We are interested in upper-bounding $X$ when $\mathbb{E}[X] \in \mathrm{O}(\log n)$. The following is a standard result.

**Corollary 2.9.** *Let $X_1, \dots, X_n$ be independent* 0–1 *random variables, $X = \sum_{i=1}^{n} X_i$, and $\mathbb{E}[X] \in \mathrm{O}(\log n)$. Then $X \in \mathrm{O}(\log n)$ w.h.p.*

*Proof.* We use that there are $c' \in \mathbb{R}_{\geq 1}$ and $n_0 \in \mathbb{N}$ such that $\mathbb{E}[X] \leq c' \log_2 n$ for all $n \geq n_0$. Given an arbitrary $c \in \mathbb{R}_{\geq 1}$, we may thus choose $R := 6cc' \log_2 n$, i.e., we have $R \geq 6\,\mathbb{E}[X]$ as well as $R \in \mathrm{O}(\log n)$, and obtain

$$\mathbb{P}[X \geq R] \overset{(2.23)}{\leq} 2^{-R} = 2^{-6cc' \log_2 n} = n^{-6cc'} \leq n^{-c}, \tag{2.24}$$

for all $n \geq n_0$, i.e., that $X \in \mathrm{O}(\log n)$ w.h.p. □

# PART I

# Metastability-Containing Circuits

*"You can't turn a no to a yes without a maybe in between."*

House of Cards, Chapter 29

This part is the result of close collaboration with Matthias Függer, Attila Kinali, and Christoph Lenzen. It is based on an article that is under submission to the *IEEE Transactions on Computers (TC)* as of May 2017, a full version is available online [55], and one that is to appear in the *IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2017),* 2017 [60]. Furthermore, a patent application has been filed [99].

The author's contribution to the paper coauthored by Attila Kinali [60] is the idea of a transistor-level CMUX, the CMUX circuits, and the improvements regarding metastability-containing sorting networks.

# CHAPTER 3
# Introduction

In digital circuits, every bistable storage element — e.g., latch or flip-flop — can become *metastable.* Metastability refers to volatile states that usually involve an internal voltage strictly between logical 0 and 1. A metastable storage element can output deteriorated signals, e.g., voltages stuck between logical 0 and logical 1, oscillations, late or unclean transitions, or show otherwise unspecified behavior. Such deteriorated signals may violate timing constraints or input specifications of gates and further storage elements. Hence, deteriorated signals may spread through combinational logic and drive further bistables into metastability. While metastability refers to a state of a bistable, we refer to the abovementioned deteriorated signals as "metastable" for the sake of exposition.

Unfortunately, any way of reading a signal from an unsynchronized clock domain or performing an analog-to-digital or time-to-digital conversion incurs the risk of a metastable result; no physical implementation of a non-trivial digital circuit can deterministically avoid, resolve, or detect metastability [108].

Traditionally, the only countermeasure is to write a potentially metastable signal into a synchronizer [13, 14, 15, 67, 88, 89] — a bistable storage element like a flip-flop — and wait. Synchronizers exponentially decrease the odds of maintained metastability over time [88, 89, 133]: In this unstable equilibrium the tiniest displacement exponentially self-amplifies and the bistable resolves metastability. Put differently, the waiting time determines the probability to resolve to logical 0 or 1. Accordingly, this approach delays subsequent computations and does not guarantee success.

We propose a fundamentally different approach: It is possible to *contain* metastability by fine-grained logical masking so that it cannot infect the entire circuit. This technique *guarantees* a limited degree of metastability in — and uncertainty about — the output. At the heart of our approach lies a model for metastability in synchronous clocked digital circuits. Metastability is propagated in a worst-case fashion, allowing to derive deterministic guarantees, without and unlike synchronizers.

**The Challenge** The problem with metastability is that it fundamentally disrupts operation in Very Large-Scale Integration (VLSI) circuits by breaking the abstraction of Boolean logic: A metastable signal can neither be viewed as being logical 0 or 1. In particular, a metastable signal is not a random bit and does not behave like an unknown but fixed Boolean signal. As an example, the circuit that computes $\neg x \vee x$ using a Not and a binary Or gate may output an arbitrary signal value if $x$ is metastable: 0, 1, or again a metastable signal. Note that this is not the case for unknown, but Boolean, $x$. The ability of such signals to "infect" an entire circuit poses a severe challenge.

**The Status Quo** The fact that metastability cannot be avoided, resolved or detected, the hazard of infecting entire circuits, and the unpleasant property of breaking the abstraction of Boolean logic have led to the predominant belief that waiting — using well-designed synchronizers — essentially is the *only* method of coping with the threat of metastability: Whenever a signal is potentially metastable, e.g., when it is communicated across a clock boundary, its value is written to a synchronizer. After a predefined time, the synchronizer output is assumed to have stabilized to logical 0 or 1, and the computation is carried out in classical Boolean logic. In essence, this approach trades synchronization delay for increased reliability; it does, however, not provide deterministic guarantees.

**Relevance** VLSI circuits grow in complexity and operating frequency, leading to a growing number unsynchronized clock domains, technology becomes smaller, and the operating voltage is decreased to save power [76]. These trends intensify the risk of metastable upsets. Treating these risks in the traditional way — by adding synchronizer stages — increases synchronization delays and thus is counterproductive w.r.t. the desire for faster systems. Hence, we urgently need alternative techniques to reliably handle metastability in both mission-critical and day-to-day systems.

**Our Approach** We challenge the point of view that synchronizers are the only way to deal with metastability and exploit that *logical masking* provides some leverage. If, e.g., one input of a NAND gate is stable 0, its output remains 1 even if its other input is arbitrarily deteriorated. This is owed to the way gates are implemented in Complementary Metal-Oxide-Semiconductor (CMOS) logic and to transistor behavior under intermediate input voltage levels.

We demonstrate that it is possible to *contain* metastability to a limited part of the circuit instead of attempting to resolve, detect, or avoid it altogether. Given Marino's result [108], this is surprising, but not a contradiction. More concretely, we show that a variety of operations can be performed in the presence of a limited degree of metastability in the input, maintaining an according guarantee on the output.

As an example, recall that in Binary Reflected Gray Code (BRGC) $x$ and $x + 1$ always only differ in exactly one bit; each upcount flips one bit. Suppose Analog-to-Digital Converters (ADCs) output BRGC but, due to their analog input, a possibly metastable bit $u$ decides whether to output $x$ or $x + 1$. As $x$ and $x + 1$ only differ in a single bit, this bit is the only one that may become metastable in an appropriate[1] implementation. Hence, all possible stabilizations are in $\{x, x + 1\}$, we refer to this as *precision*-1. Among other things, we show that it is possible to sort such inputs in a way that the output still has precision-1.

We assume worst-case metastability propagation and still are able to *guarantee* correct results. This opens up an alternative to the classic approach of *postponing* the computation by first using synchronizers. Advantages over synchronizers are:

(1) No time is lost waiting for (possible) stabilization. This permits fast response times as, e.g., useful for high-frequency clock synchronization in hardware, see Chapter 10. Note that this removes synchronization delay from the list of fundamental limits to the operating frequency.

(2) Correctness is guaranteed deterministically instead of probabilistically.

[1]The appropriate implementation is a metastability-containing multiplexer with select bit $u$ and inputs $x$ and $x + 1$. We discuss this in Chapter 6.

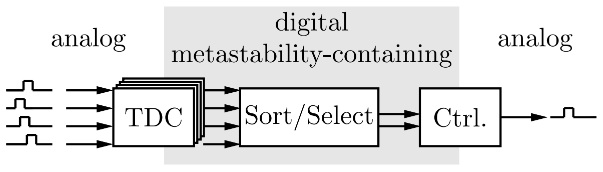


**Figure 3.1:** The separation of concerns (analog – digital metastability-containing – analog) for fault-tolerant clock synchronization in hardware.

(3) Stabilization can, but is not required to, happen "during" the computation, i.e., synchronization and calculation happen simultaneously.

**Separation of Concerns** Clearly, the impossibility of deterministically resolving metastability [108] still holds; metastability may still occur, even if it is contained. Hence, a *separation of concerns,* compare Figure 3.1, is key to our approach.

For the purpose of illustration, consider a hardware clock-synchronization algorithm, we discuss this in Chapter 10. We start in the *analog* world: nodes generate clock pulses. Each node measures the time differences between its own and all other nodes' pulses using Time-to-Digital Converters (TDCs). Since this involves entering the *digital* world, metastability in the measurements is unavoidable [108]. The traditional approach is to hold the TDC outputs in synchronizers, spending time and thus imposing a limit on the operating frequency. But as discussed above, it is possible to limit the metastability of each measurement to at most one bit in BRGC-encoded numbers, where the metastable bit represents the "uncertainty between $x$ and $x + 1$ clock ticks," i.e., precision-1.

We apply *metastability-containing* components to digitally process these inputs to derive digital correction parameters for the node's oscillator. These parameters contain at most one metastable bit, as above accounting for precision-1. We convert them to an *analog* control signal for the oscillator. This way, the metastability translates to a small frequency offset within the uncertainty from the initial TDC measurements.

In short, metastability is introduced at the TDCs, *deterministically* contained in the digital subcircuit, and ultimately absorbed by the analog control signal.

## 3.1 Our Contribution

In detail, our contributions are the following.

(1) In Chapter 4, we present a rigorous time-discrete value-discrete model for metastability in clocked as well as in purely combinational digital circuits. We consider two types of registers: simple (standard) registers that do not provide any guarantees regarding metastability and masking registers that can "hide" internal metastability to some degree using high- or low-threshold inverters. The propagation of metastability is modeled in a worst-case fashion and metastable registers may or may not stabilize to 0 or 1. Hence, the resulting model allows us to derive deterministic guarantees concerning circuit behavior under metastable inputs.

We consider the model that allows a novel and fundamentally different worst-case treatment of metastability our main contribution. Accordingly, we are obligated to verify that it properly reflects the physical behavior of digital circuits, i.e., that it is sufficiently pessimistic. At the same time, the model is useless if it is too pessimistic in the sense that it does not allow non-trivial positive results. We address these points in Chapters 5 and 6.

(2) We perform a reality check, showing in Chapter 5 that the physical impossibility of avoiding, resolving, or detecting metastability [108] holds in our model. Any model dealing with metastability must reflect these properties.

(3) In Chapter 6, we turn our attention to Multiplexers (MUXes); they constitute a good example of the problems caused by ignoring metastability. We fix these issues by developing Metastability-Containing Multiplexers (CMUXes). This illustrates what we mean by "containing metastability" in the light of the impossibility of resolving it, shows that our model permits non-trivial positive results, and demonstrates key aspects of our model, for example the spread of metastability through the combinational logic, clocked and unclocked circuits, and how to use masking registers.

    Furthermore, we develop transistor-level implementations of CMUXes. These are not covered by our gate-level model, but are relevant w.r.t. practical implementations and show how to apply our line of reasoning to CMOS logic. The practical relevance of transistor-level CMUXes is demonstrated by massively reducing the size of existing metastability-containing sorting networks.

Having established some confidence that our model properly reflects the physical world and allows reasoning about circuit design, we turn our attention to the question of computability.

(4) We analyze the computational power of *combinational circuits* in Chapter 7. Combinational circuits are a special class of circuits that essentially only comprise combinational logic. We show that they behave "as expected," e.g., that additional clock cycles do not enhance their computational power.

(5) Chapter 8 constitutes a core contribution. We analyze what functions are computable by circuits w.r.t. the available register types, the number of clock cycles, and whether a circuit is combinational. Let $\mathrm{Fun}_M^r$ denote the class of functions that can be implemented by an arbitrary circuit in $r$ clock cycles;[2] analogously, let $\mathrm{Fun}_S^r$ and $\mathrm{Fun}_C^r$ denote the classes of functions implementable in $r$ clock cycles of circuits that can only use simple registers, *simple circuits,* and by combinational circuits, respectively.

    We show that the number of clock cycles is irrelevant for combinational and simple circuits: $\mathrm{Fun}_C := \mathrm{Fun}_C^1 = \mathrm{Fun}_C^r$ and $\mathrm{Fun}_S := \mathrm{Fun}_S^1 = \mathrm{Fun}_S^r$ for all $r \in \mathbb{N}$. This reflects the intuition from electrical engineering that synchronous Boolean circuits can be unrolled. It is, however, not obvious that unrolling works in the presence of metastability; hence we establish these claims.

[2] The $M$ indicates that the circuit may comprise masking registers.

The abovementioned non-obviousness is reflected in a surprising result: In the presence masking registers, unrolling does not yield equivalent circuits and we obtain a strict inclusion:

$$\mathrm{Fun}_C = \mathrm{Fun}_S = \mathrm{Fun}_M^1 \subsetneq \mathrm{Fun}_M^2 \subsetneq \mathrm{Fun}_M^3 \subsetneq \cdots . \tag{3.1}$$

(6) In Chapter 9, we move on to demonstrating that with combinational and simple circuits, many non-trivial functions can be computed in the face of worst-case propagation of metastability. To this end, we fully classify $\mathrm{Fun}_C = \mathrm{Fun}_S$, the class of functions that can be computed by such circuits. Furthermore, we establish the *metastable closure,* the strictest possible extension of a function specification that allows it to be computed by a combinational or simple circuit. Our classification provides a simple test for deciding whether a desired specification can be implemented by such circuits.

Finally, we apply our techniques to show that an advanced, useful circuit is in reach.

(7) We show in Chapter 10 that all arithmetic operations required by the widely used [17, 92, 93, 94] fault-tolerant clock synchronization algorithm of Lundelius Welch and Lynch [106] — max and min, sorting, and conversion between Thermometer Code (TC) and BRGC — can be performed in a metastability-containing manner. Employing the abovementioned separation of concerns, a hardware implementation of the entire algorithm is within reach, providing the deterministic guarantee that the algorithm works correctly at all times, despite metastable upsets originating in the TDCs and without synchronizers.

Note that the algorithm tolerates $f < n/3$ Byzantine faults — Byzantine nodes can show arbitrarily malicious behavior [96] — but under the assumption that nodes adhere to Boolean logic. Metastable upsets, however, are harsher errors: A metastable measurement may behave inconsistently *within a node,* violating the abstraction of Boolean logic as indicated above, which is different from receiving a faulty but stable message.

As a consequence, we show that (a) synchronization delay poses no fundamental limit on the operating frequency of clock synchronization in hardware and that (b) clock domains can be synchronized without synchronizers. The latter shows that we may eliminate communication across unsynchronized clock domains as a source of metastable upsets altogether.

We conclude this part in Chapter 11.

## 3.2 Related Work

**Metastability** The phenomenon of metastable signals has been studied for decades [88] with the following key results. (1) No physical implementation of a digital circuit can reliably avoid, resolve, or detect metastability; any digital circuit, including "detectors," producing different outputs for different input signals can be forced into metastability [108]. (2) The probability of an individual event generating metastability can be kept low. Large transistor counts and high operational frequencies, low supply voltages, temperature effects, and trends in technology, however, disallow to neglect the problem [14]. (3) Being an unstable equilibrium, the probability that, e.g., a memory cell remains in a metastable state decreases exponentially over time [88, 89, 133]. Thus, waiting for a sufficiently long time reduces the probability of sustained metastability to within acceptable bounds.

**Synchronizers** The predominant technique to cope with metastable upsets is to use synchronizers [13, 14, 15, 67, 88, 89], carefully designed [13, 67] bistable storage elements that hold potentially metastable signals, e.g., after communicating them across a clock boundary. After a predefined time, the synchronizer output is assumed to have stabilized to logical 0 or 1 and the computation is carried out in classical Boolean logic. In essence, this approach trades delay for increased reliability, typically expressed as Mean Time Between Failures (MTBF)

$$\text{MTBF} = \frac{e^{t/\tau}}{T_W F_C F_D}, \tag{3.2}$$

where $F_C$ and $F_D$ are the clock and data transition frequencies, $\tau$ and $T_W$ are technology-dependent values, and $t$ is the predetermined time allotted for synchronization [13, 14, 15, 67, 88, 89, 129, 130] referred to as *synchronization delay.* Note that classical bounds for the MTBF assume a uniform distribution of clock and data transitions [13, 89]. Synchronizers, however, do not provide deterministic guarantees and avoiding synchronization delay is an important issue [129, 130].

**Glitch/Hazard Propagation** Metastability-containing circuits are related to glitch-free/hazard-free circuits, which have been extensively studied since Huffman [74] and Unger [132] introduced them. Eichelberger [43] extended these results to multiple switching inputs and dynamic hazards, Brzozowski and Yoeli extended the simulation algorithm [25], Brzozowski et al. surveyed techniques using higher-valued logics [24] such as Kleene's 3-valued extension of Boolean logic [90], and Mendler et al. studied delay requirements needed to achieve consistency with simulated results [111].

While we too resort to Kleene logic to model metastability, there are differences to the classical work on hazard-tolerant circuits: (1) A common assumption in hazard detection is that inputs only perform well-defined, clean transitions, i.e., the assumption of a hazard-free input-generating circuitry is made. This is the key difference to metastability-containment: Metastability encompasses much more than inputs that are in the process of switching; metastable signals may or may not be in the process of completing a transition, may be oscillating, and may get "stuck" at an intermediate voltage. (2) Another common assumption in hazard detection is that circuits have a constant delay. This is no longer the case in the presence of metastability; unless metastability is properly masked, circuit delays can deteriorate in the presence of metastable input signals, even if the circuit eventually generates a stable output [60]. This can cause late transitions that potentially drive further registers into metastability. (3) Glitch-freedom is no requirement for metastability-containment. (4) When studying synthesis, we allow for specifications where outputs may contain metastable bits. This is necessary for non-trivial specifications in the presence of metastable inputs [108]. (5) We allow a circuit to compute a function in multiple clock cycles. (6) Circuits may comprise masking registers [88].

**OR Causality** The work on weak (OR) causality in asynchronous circuits [134] studies the computation of functions under availability of only a proper subset of its parameters. As an example, consider a Boolean function $f(x, y)$, where $f(0, 0) = f(0, 1)$. An early-deciding asynchronous module may set its output as soon as $x = 0$ arrives at its input, disregarding the value of $y$. Early-deciding circuits, however, differ from our work because they are neither clocked synchronous designs nor do they necessarily operate correctly in presence of metastable input bits: $f(0, \mathrm{M}) = f(0, 0) = f(0, 1)$ does not necessarily hold.

**Speculative Computing** To the best of our knowledge, the most closely related work is that by Tarawneh et al. on speculative computing [129, 130]. The idea is the following: When computing $f(x, y)$ in presence of a potentially metastable input bit $x$, (1) speculatively compute both $f(0, y)$ and $f(1, y)$, (2) in parallel, store the input bit $x$ in a synchronizer for a predefined time that provides a sufficiently large probability of resolving metastability of $x$, and (3) use $x$ to select whether to output $f(0, y)$ or $f(1, y)$. This hides (a part of) the synchronization delay allotted to $x$.

Like our approach, speculative computations allow for an overlap of synchronization and computation time. The key differences are: (1) Relying on synchronizers, speculative computing incurs a non-zero probability of failure; metastability-containment insists on deterministic guarantees. (2) In speculative computing, the set of potentially metastable bits $X$ must be known in advance. Regardless of the considered function, the complexity of a speculative circuit grows exponentially in $|X|$. Neither is the case for metastability–containment, as illustrated by several circuits [26, 62, 100, 128]. (3) Our model is rooted in an extension of Boolean logic, i.e., uses a different function space. Hence, we face the question of computability of such functions by digital circuits; this question does not apply to speculative computing as it uses traditional Boolean functions.

**Metastability-Containing Circuits** Since the foundation of this part was first published [55], many of the proposed the techniques have been successfully employed to obtain metastability-aware TDCs [62], metastability-containing BRGC sorting networks [26, 100], CMUXes [60],[3] and metastability-tolerant network-on-chip routers [128]. Simulations verify the positive impact of metastability-containing techniques [26, 60, 128]. Most of these works channel efforts towards metastability-containing Field-Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) implementations of fault-tolerant distributed clock synchronization; this part establishes that all required components are within reach.

## 3.3 Notation and Preliminaries

We abbreviate $[k] := \{\ell \in \mathbb{N}_0 \mid \ell < k\}$ for $k \in \mathbb{N}_0$ and concatenate tuples using

$$\circ\colon (A_1 \times \cdots \times A_m) \times (B_1 \times \cdots \times B_n) \to (A_1 \times \cdots \times A_m \times B_1 \times \cdots \times B_n) \tag{3.3}$$

$$(a_1, \ldots, a_m) \circ (b_1, \ldots, b_n) := (a_1, \ldots, a_m, b_1, \ldots, b_n). \tag{3.4}$$

[3]We present the parts of this paper the author has contributed. Whenever referring to further results from this work — e.g., simulation results — we cite it as related work.

For functions $f, g\colon X \to \mathcal{P}(Y)$, we call $g$ a *subfunction of* $f$ and write $g \subseteq f$ if and only if $g(x) \subseteq f(x)$ for all $x \in X$.

# CHAPTER 4
# Circuit Model

We propose a discrete-time and discrete-value circuit model in which registers can become metastable and their resulting output signals deteriorated. The model supports synchronous, clocked circuits composed of registers and combinational logic as well as purely combinational circuits. Specifically, we study the generic synchronous state-machine design depicted in Figure 4.1. Data is initially available in the input registers. At each rising clock transition, local and output registers update their state according to the circuit's combinational logic. Figure 4.1(b) shows the phases of a clock cycle:

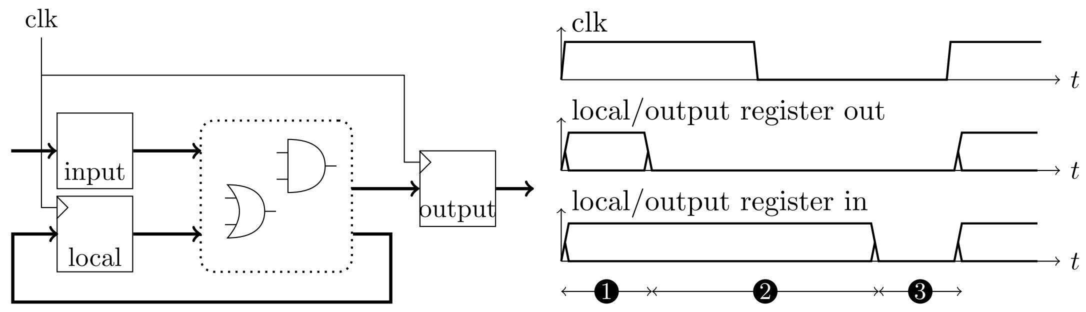
(1) In the first phase, the output of the recently updated local and output registers stabilizes. This is accounted for by the *clock-to-output* time that is bounded, unless a register is metastable; in this case, no deterministic upper bound exists.

(2) During phase two, the register output propagates through the combinational logic to the register inputs. Its duration can be upper-bounded by the worst-case propagation delay through the combinational logic. If a register output is deteriorated, however, this is unbounded as well, unless the according signals are properly masked.

(3) In the third phase, the register inputs are stable, ready to be sampled, and result in updated local and output register states. The duration of this phase can account for small extra delays in phase (1), possibly mitigating some metastable upsets. If phase (1) or (2), however, take too long due to metastability and the corresponding deteriorated signals, registers may read a deteriorated input value, potentially driving the register into metastability.

As motivated in Chapter 3, metastable registers output an unspecified, arbitrarily deteriorated signal. Deteriorated can mean any constant voltage between logical 0 and logical 1, arbitrary signal behavior over time, oscillations, or simply violated timing constraints, such as late signal transitions. Furthermore, deteriorated signals can cause registers to become metastable, e.g., due to violated constraints regarding timing or input voltage. Knowing full well that metastability is a state of a *bistable* element and not a signal value or voltage, we still need to talk about the "deterioration caused by or potentially causing metastability in a register" in *signals.* For the sake of presentation — and as these effects are causally linked — we refer to both phenomena using the term *metastability* without making the distinction explicit.

Our model uses Kleene's 3-valued logic [90], a ternary extension of Boolean logic. The third value of the Kleene logic appropriately expresses the uncertainty about gate behavior in the presence of metastability. In the absence of metastability our model behaves like a traditional, deterministic, binary circuit model. In order to obtain *deterministic* guarantees, we assume worst-case propagation of metastability: If a signal can be "infected" by metastability, there is no way to prevent it.



**(a)** Synchronous circuit. **(b)** Three-phase clock cycle.

**Figure 4.1:** Generic synchronous state machine (a) and clock cycle (b). Input registers are prefilled, local and output registers are updated at each rising clock transition. Clock cycles comprise three phases: (1) register-output stabilization, (2) propagation of outputs through combinational logic to register inputs, and (3) stable register inputs.

Given the elusive nature of metastability, it is easy to jump to conclusions. Hence, we show in Chapter 5 that the proposed model reproduces well-known impossibility results from the realm of digital circuit design, namely, the impossibility of avoiding, detecting, and resolving metastability in physical implementations of digital circuits established by Marino [108]. This obliges us to provide evidence that our model has practical relevance, i.e., that it is indeed possible to perform meaningful computations. We meet this obligation in Chapter 6, where we develop CMUXes. Together with recent results complementing ours [26, 62, 100, 128], we are confident that our model makes meaningful predictions.

In our model, circuits are synchronous state machines: Combinational logic, represented by gates, maps a circuit state to possible successor states, compare Figure 4.1. The combinational logic uses, and registers store, *signal values* $\mathbb{B}_\mathrm{M} := \{0, 1, \mathrm{M}\}$. M represents a *metastable* register state or — w.r.t. the simplification in terminology motivated above — signal value. Metastability is the only source of non-determinism in our model. The classical Boolean signal values are $\mathbb{B} = \{0, 1\}$. Let $x \in \mathbb{B}_\mathrm{M}^k$ be a $k$-bit tuple. We call $x$ *stable* if and only if $x \in \mathbb{B}^k$. Stored in registers over time, metastable bits may resolve to 0 or 1. The set of partial resolutions of $x$ is $\mathrm{Res}_\mathrm{M}(x)$ and the set of metastability-free, completely stabilized, resolutions is $\mathrm{Res}(x)$. If $m$ bits in $x$ are metastable, $|\mathrm{Res}_\mathrm{M}(x)| = 3^m$ and $|\mathrm{Res}(x)| = 2^m$, since M serves as "wildcard" for $\mathbb{B}_\mathrm{M}$ and $\mathbb{B}$, respectively. Formally, we have

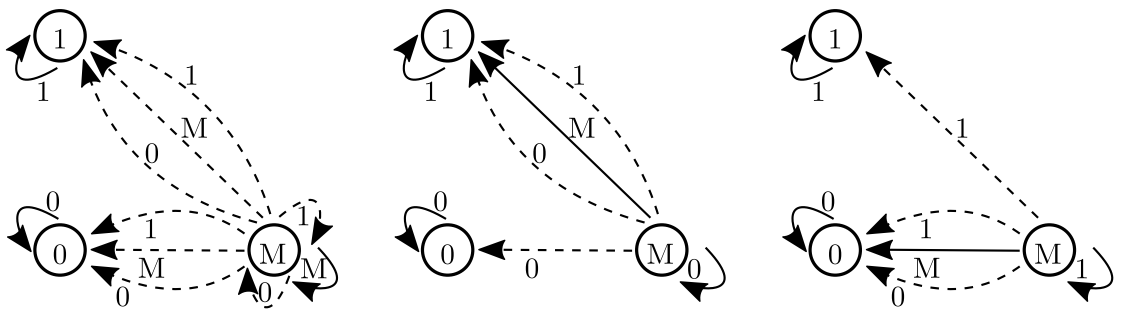
$$\mathrm{Res}_\mathrm{M}(x) := \left\{ y \in \mathbb{B}_\mathrm{M}^k \mid \forall i \in [k] \colon x_i = y_i \vee x_i = \mathrm{M} \right\} \text{ and} \tag{4.1}$$

$$\mathrm{Res}(x) := \mathrm{Res}_\mathrm{M}(x) \cap \mathbb{B}^k. \tag{4.2}$$

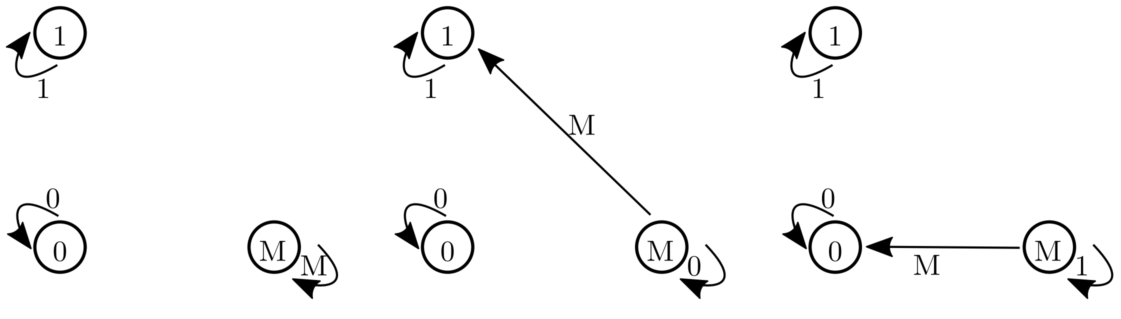
The *metastable superposition* [100] captures the intuition of "overlaying" signals:

$$\oplus \colon \mathbb{B}_\mathrm{M}^k \times \mathbb{B}_\mathrm{M}^k \to \mathbb{B}_\mathrm{M}^k \tag{4.3}$$

$$(x \oplus y)_i := \begin{cases} x_i & \text{if } x_i = y_i \text{ and} \\ \mathrm{M} & \text{otherwise.} \end{cases} \tag{4.4}$$



**(a)** Simple register, model of physical behavior. **(b)** Mask-0 register, model of physical behavior. **(c)** Mask-1 register, model of physical behavior.



**(d)** Simple register, worst-case behavior, used below. **(e)** Mask-0 register, worst-case behavior, used below. **(f)** Mask-1 register, worst-case behavior, used below.

**Figure 4.2:** We model registers as non-deterministic state machines. State transitions represent a clock cycle elapsing and are labeled with what is read (sampled) at a register's output at the according rising clock flank. Figures (a)–(c) model the physical register behavior. Assuming worst-case behavior, we may simplify the state machines by omitting the dashed lines and obtain Figures (d)–(f), which we use in our model.

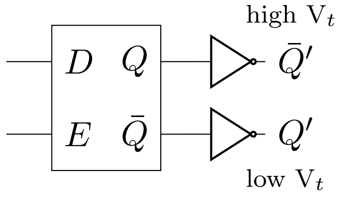
Clearly, $(\mathbb{B}_{\mathrm{M}}^k, \oplus)$ is a commutative semigroup. Hence, we may abbreviate $\bigoplus_{x \in X} x$ for non-empty[1] $X \subseteq \mathbb{B}_{\mathrm{M}}^k$.

## 4.1 Registers

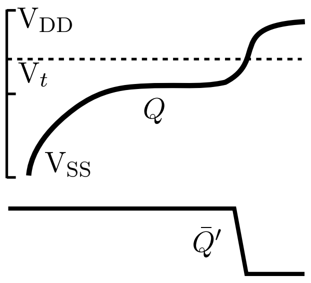
The phenomenon of metastability is intrinsically linked to time: It is a volatile state that quickly decays; the odds of maintained metastability decrease exponentially over time [88, 89, 133], which is exploited in synchronizers that achieve an MTBF which exponentially increases with the time allotted for synchronization [13, 14, 15, 67, 88, 89]. However, we propose a time-discrete model for metastability. Hence, correctly modeling registers is the critical step, as they are the only elements in our model that permit maintaining a state across clock cycles, i.e., over time.

To this end, we proceed in two steps. The first is to model the physical register behavior with all degrees of freedom incurred by metastability. We do this using state

[1] $X$ may not be empty because $(\mathbb{B}_{\mathrm{M}}^k, \oplus)$ has no neutral element. This makes sense, since the superposition of an empty set of signals possesses no meaningful interpretation in this context.



(a) Latch with high-threshold inverter to obtain a mask-0 register.



(b) The high-threshold inverter masks deteriorated signals.

**Figure 4.3:** Masking registers can, e.g., be implemented by attaching high- or low threshold inverters to a flip-flop's slave latch (a). This amplifies internal metastability to a stable output (b). It is still possible to observe a deteriorated output while the signal crosses the threshold. As the signal is already being amplified at this point, this time window is short. Figure adapted from Section 3.1 of Kinniment [88].

machines, where the state transitions indicate the possible behavior within one clock cycle; this is depicted in Figure 4.2(a)–4.2(c). This leaves us with rather complex state machines that are not amenable to a theoretical analysis. Fortunately, we benefit from proposing a worst-case model, where we may make pessimistic simplifications like "if the register may be read as metastable or stable we assume it is read as M." This is reflected in the state machines depicted in Figures 4.2(d)–4.2(f); they greatly simplify the analysis and we use them throughout this part.

We consider three types of single-bit registers, all of which behave just like in binary circuit models unless metastability occurs:

**simple** registers are oblivious to metastability in the sense that they provide no guarantees regarding their behavior once they are metastable. The physical behavior is captured by the state machine in Figure 4.2(a), we use the pessimistic simplification in Figure 4.2(d), see below.

**mask-0** registers output 0 while in state M. If they stabilize to state 0, the intermediate metastability is never observed at the output. But if they stabilize to 1, there is a brief time window, in our case one clock cycle, when metastability is propagated to the output.

Physical realizations of masking registers are obtained by flip-flops with high-threshold inverters at the output, amplifying an internal metastable signal to 0; see, e.g., Section 3.1 on metastability filters of Kinniment [88]. This effectively "hides" metastability below the inverter threshold, see Figure 4.3. However, it is still possible to observe a deteriorated signal while a mask-0 register crosses the inverter threshold, hence the one round in which M can be observed. A mask-0 register behaves according to the state machine in Figure 4.2(b) which simplifies to that in Figure 4.2(e) as argued below.

**mask-1** registers are analogous to mask-0 registers. The complex behavior is depicted in Figure 4.2(c) and simplifies to that in Figure 4.2(f).

In our model, a register $R$ has a *type* (simple, mask-0, or mask-1) and a state $x_R \in \mathbb{B}_{\mathrm{M}}$. $R$ behaves according to $x_R$ and its type's non-deterministic state machine in Figures 4.2(d)–4.2(f). Each clock cycle, $R$ performs one state transition annotated with some $o_R \in \mathbb{B}_{\mathrm{M}}$, which is the result of sampling $R$ at that clock cycle's rising clock flank. This happens exactly once per clock cycle and we refer to it as *reading R.* The state transitions account for the possible resolution of metastability over time.

Consider the simple register in Figure 4.2(a). When in state 0, its output and successor state are both 0; it behaves symmetrically in state 1. In state M, however, any output in $\mathbb{B}_{\mathrm{M}}$ combined with any successor state in $\mathbb{B}_{\mathrm{M}}$ is possible. Since our goal is the design of circuits that operate correctly under metastability even if it never resolves, we make two pessimistic simplifications:

(1) First, if there are three parallel state transitions from state M to $x$ with outputs $0, 1, \mathrm{M}$, we only keep the one with output M and then

(2) if, for some fixed output $o \in \mathbb{B}_{\mathrm{M}}$, there are state transitions from a state M to multiple states including M, we only keep the one with successor state M. This maintains the possibility of encountering metastability in future clock cycles.

These simplifications are obtained by ignoring the dashed state transitions in Figures 4.2(a)–4.2(c) and yield the state machines in Figures 4.2(d)–4.2(f). In the following, we only use the state machines in Figures 4.2(d)–4.2(f). Observe that the dashed lines are an artifact of the highly non-deterministic behavior in the physical world. If one is pessimistic about the behavior — which we are — one obtains the proposed simplification.

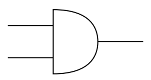
Regarding simple registers, the above simplifications yield a deterministic state machine. The mask-$b$ registers, $b \in \mathbb{B}$, shown in Figures 4.2(e) and 4.2(f), exhibit the following behavior: As long as their state remains M, they output $b \neq \mathrm{M}$; only when their state changes from M to $1-b$ they output M once, after that they are stable.

## 4.2 Gates

We model the behavior of combinational gates in the presence of metastability. A *gate* is defined by $k \in \mathbb{N}_0$ input ports, one output port — gates with $k \geq 2$ distinct output ports are represented by $k$ single-output gates — and a Boolean function $f\colon \mathbb{B}^k \to \mathbb{B}$. We generalize $f$ to $f_{\mathrm{M}}\colon \mathbb{B}_{\mathrm{M}}^k \to \mathbb{B}_{\mathrm{M}}$ by

$$f_{\mathrm{M}}(x) := \bigoplus_{x' \in \mathrm{Res}(x)} f(x') = \begin{cases} 0 & \text{if } f(x') = 0 \text{ for all } x' \in \mathrm{Res}(x), \\ 1 & \text{if } f(x') = 1 \text{ for all } x' \in \mathrm{Res}(x), \text{ and} \\ \mathrm{M} & \text{otherwise.} \end{cases} \tag{4.5}$$

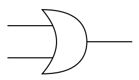
The premise leading to this definition is that each metastable input can be badly deteriorated but might also "look like" a 0 or 1. Hence, if all stabilizations $x' \in \mathrm{Res}(x)$ lead to the same output $f(x') = b \neq \mathrm{M}$, the metastable bits in $x$ have no influence on the



| $f^{\textsc{And}}$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| $f_{\mathrm{M}}^{\textsc{And}}$ | 0 | 1 | M |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | M |
| M | 0 | M | M |

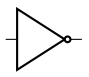
**(a)** And.



| $f^{\textsc{Or}}$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| $f_{\mathrm{M}}^{\textsc{Or}}$ | 0 | 1 | M |
|---|---|---|---|
| 0 | 0 | 1 | M |
| 1 | 1 | 1 | 1 |
| M | M | 1 | M |

**(b)** Or.



| $f^{\textsc{Not}}$ | – |
|---|---|
| 0 | 1 |
| 1 | 0 |

| $f_{\mathrm{M}}^{\textsc{Not}}$ | – |
|---|---|
| 0 | 1 |
| 1 | 0 |
| M | M |

**(c)** Not.

**Table 4.1:** Gate behavior under metastability. We extend $f\colon \mathbb{B}^k \to \mathbb{B}$ to $f_{\mathrm{M}}\colon \mathbb{B}_{\mathrm{M}}^k \to \mathbb{B}_{\mathrm{M}}$ such that $f_{\mathrm{M}}(x) = b \neq \mathrm{M}$ if and only if $f$ returns the same value for all full stabilizations of $x$, otherwise $f_{\mathrm{M}}(x) = \mathrm{M}$. The top row depicts gate symbols.

output of the gate.[2] Otherwise, the metastable bits do have an influence on $f(x)$ and the gate can output a metastable signal. Whenever a gate can output a metastable signal, we assume it does, reflecting our goal of devising a worst-case model. Observe that, due to $\mathrm{Res}(x) = \{x\}$ for all stable $x \in \mathbb{B}^k$, we have

$$\forall x \in \mathbb{B}^k\colon \quad f_{\mathrm{M}}(x) = f(x) \in \mathbb{B}. \tag{4.6}$$

Hence, gate behavior is compatible with classical Boolean gates if no metastability occurs.

Our definitions are equivalent to Kleene logic [90], see Table 4.1. Note that Kleene logic has been used by, e.g., Eichelberger [43] to model hazards, where the third value represents switching signals. As motivated above, our definition of "M" includes arbitrary signal behavior over time, i.e., includes switching signals as well. Hence, it seems natural that we use Kleene logic as well.

As an example, consider Table 4.1(a) and the And gate with two input ports implementing $f^{\textsc{And}}(x_1, x_2) = x_1 \wedge x_2$. We extend $f^{\textsc{And}}\colon \mathbb{B}^2 \to \mathbb{B}$ to $f_{\mathrm{M}}^{\textsc{And}}\colon \mathbb{B}_{\mathrm{M}}^2 \to \mathbb{B}_{\mathrm{M}}$. For $x \in \mathbb{B}^2$, we have $f^{\textsc{And}}(x) = f_{\mathrm{M}}^{\textsc{And}}(x)$. Next, consider $x = \mathrm{M}1$. As $\mathrm{Res}(\mathrm{M}1) = \{01, 11\}$, we have $f_{\mathrm{M}}^{\textsc{And}}(\mathrm{M}1) = f^{\textsc{And}}(01) \oplus f^{\textsc{And}}(11) = 0 \oplus 1 = \mathrm{M}$. For $x = \mathrm{M}0$, on the other hand, we obtain $\mathrm{Res}(x) = \{00, 10\}$ and $f_{\mathrm{M}}^{\textsc{And}}(\mathrm{M}0) = f^{\textsc{And}}(00) \oplus f^{\textsc{And}}(10) = 0 \oplus 0 = 0$, i.e., the metastable bit is *masked.*

The Not, Or, and other gates are handled analogously. Refer to Chapter 6 and to Figure 6.2 in particular for an example of metastability in combinational logic.

## 4.3 Combinational Logic

We model combinational logic as Directed Acyclic Graph (DAG) $G = (V, A)$ with parallel arcs, compare Figure 4.4. Each node either is an *input node,* an *output node,* or a gate

[2] This holds for standard CMOS implementations of common gates like Not, Or, And, Nor, and Nand. However, it has to be checked with care w.r.t. the underlying technology; possibly, one has to work with a subset of the gates or modify some of them. Our transistor-level CMUX implementation in Chapter 6 can be interpreted as "implementing a MUX gate."

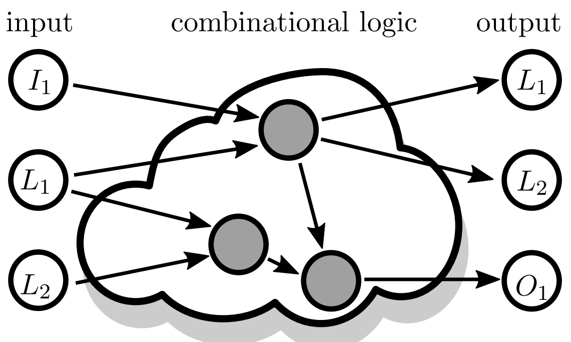


**Figure 4.4:** Combinational logic DAG with gates (gray) and registers (white). The input ($I_1$), output ($O_1$), and local ($L_1$ and $L_2$) registers occur as input nodes, output nodes, and both, respectively.

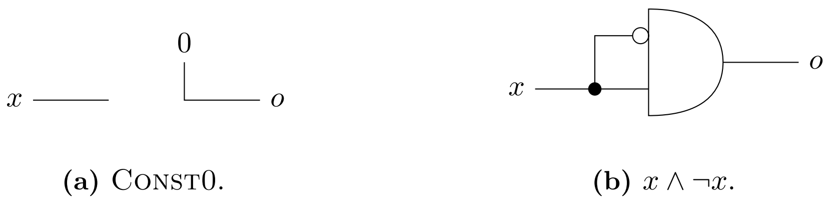


**Figure 4.5:** Two combinational logic DAGs implementing $x \mapsto 0$ (a) and $x \mapsto x \land \neg x$ (b). While these are equivalent w.r.t. Boolean logic, i.e., under stable inputs, they are not equivalent w.r.t. Kleene logic, i.e., under metastable inputs.

(see Section 4.2).

Input nodes are sources in the DAG, i.e., have indegree 0 and an arbitrary outdegree, and output nodes are sinks with indegree 1, i.e., have indegree 1 and outdegree 0. If $v \in V$ is a gate, denote by $f_v \colon \mathbb{B}_{\mathrm{M}}^{k_v} \to \mathbb{B}_{\mathrm{M}}$ its gate function — generalized to possibly metastable inputs and outputs as in Section 4.2 — with $k_v \in \mathbb{N}_0$ parameters. For each parameter of $f_v$, $v$ is connected to exactly one input node or gate $w$ by an arc $(w, v) \in A$. Every output node $v$ is connected to exactly one input node or gate $w$ by an arc $(w, v) \in A$.

Note that input nodes and gates can serve as input to multiple gates and output nodes, i.e., the fan-out is unbounded. Furthermore, observe that $G$ may have sources that are no input nodes; these are gates without input and with constant output, i.e., Const0 and Const1. Lastly, note that $G$ may comprise sinks that do not correspond to output nodes if the output of a gate is ignored.

Suppose $G$ has $m$ input nodes and $n$ output nodes. Then $G$ defines a function $f^G \colon \mathbb{B}_{\mathrm{M}}^m \to \mathbb{B}_{\mathrm{M}}^n$ as follows. Starting with input $x \in \mathbb{B}_{\mathrm{M}}^m$, we evaluate the nodes $v \in V$. If $v$ is an input node, it evaluates to $x_v$. Gates of indegree 0 are constants and evaluate accordingly. If $v$ is a gate of non-zero indegree, it evaluates to $f_v(\bar{x})$, where $\bar{x} \in \mathbb{B}_{\mathrm{M}}^{k_v}$ is the recursive evaluation of all nodes $w$ with $(w, v) \in A$. Otherwise, $v$ is an output node, has indegree 1, and evaluates just as the unique node $w$ with $(w, v) \in A$. Finally, $f^G(x)_v$ is the evaluation of the output node $v$.

**Observation 4.1.** *Carefully note that two combinational logic DAGs that behave iden-*

*tically under stable inputs may show different behavior in the presence of metastability. This is, in fact, a major point about containing metastability. As an example consider DAGs $G$ and $H$ with one input and one output node in Figure 4.5. $G$ in Figure 4.5(a) ignores the input and always outputs* 0 *using a* Const0 *gate. The DAG $H$, see Figure 4.5(b), uses a* Not *and an* And *gate to output $x \wedge \neg x$, where $x$ is the input. We obtain $f^G, f^H : \mathbb{B}_\mathrm{M} \to \mathbb{B}_\mathrm{M}$ with*

$$f^G(0) = f^G(1) = 0 = f^H(0) = f^H(1). \tag{4.7}$$

*However, we have*

$$f^G(\mathrm{M}) = 0 \neq \mathrm{M} = f^H(\mathrm{M}). \tag{4.8}$$

*For a more relevant example refer to the CMUXes in Chapter 6, in particular to Figure 6.2.*

We proceed with a fundamental lemma about the behavior of combinational logic DAGs. Provided with a partially metastable input $x$, some gates — those where the collective metastable input ports have an impact on the output — evaluate to M. So when stabilizing $x$ bit by bit, no new metastability is introduced at the gates. Furthermore, once a gate stabilized, its output is fixed; further stabilizing the input does not lead to destabilizing the output. Formally, we obtain the following lemma.

**Lemma 4.2.** *Let $G$ be a combinational logic DAG with $m$ input nodes. Then for all $x \in \mathbb{B}_\mathrm{M}^m$,*

$$x' \in \mathrm{Res}_\mathrm{M}(x) \quad \Rightarrow \quad f^G(x') \in \mathrm{Res}_\mathrm{M}\left(f^G(x)\right). \tag{4.9}$$

*Proof.* We show the statement by induction on $|V|$. For the sake of the proof we extend $f^G$ to all nodes of $G = (V, A)$, i.e., write $f^G(x)_v$ for the evaluation of $v \in V$ w.r.t. input $x$, regardless of whether $v$ is an output node. The claim is trivial for $|V| = 0$. Hence, suppose the claim holds for DAGs with up to $i \in \mathbb{N}_0$ vertices, and consider a DAG $G = (V, A)$ with $|V| = i+1$. As $G$ is non-empty, it contains a sink $v \in V$. Removing $v$ allows applying the induction hypothesis to the remaining graph, proving that $f^G(x')_w \in \mathrm{Res}_\mathrm{M}(f^G(x)_w)$ for all nodes $w \neq v$.

Concerning $v$, the claim is immediate if $v$ is a source, because $f^G(x)_v = x_v$ if $v$ is an input node and $f^G(x)_v = b$ for a constant $b \in \mathbb{B}_\mathrm{M}$ if $v$ is a gate of indegree 0. If $v$ is an output node, it evaluates to the same value as the unique node $w$ with $(w, v) \in A$, which behaves as claimed by the induction hypothesis. Otherwise $v$ is a gate of non-zero indegree; consider the nodes $w \in V$ with $(w, v) \in A$. For input $x$, $v$ receives input $\bar{x} \in \mathbb{B}_\mathrm{M}^{k_v}$, whose components are given by $f^G(x)_w$; define $\bar{x}'$ analogously w.r.t. input $x'$. By the induction hypothesis, we have $\bar{x}' \in \mathrm{Res}_\mathrm{M}(\bar{x})$. If $f_v(\bar{x}) = \mathrm{M}$, the claim holds because $\mathrm{Res}_\mathrm{M}(\mathrm{M}) = \mathbb{B}_\mathrm{M}$. On the other hand, for the case that $f_v(\bar{x}) = b \neq \mathrm{M}$, our gate definition entails that $f_v(\bar{x}') = b$, because $\bar{x}' \in \mathrm{Res}_\mathrm{M}(\bar{x})$. □

Note that the evaluation of the combinational logic is entirely deterministic in our model. In the physical world, however, the amplification of digital gates may "stabilize" a deteriorated input signal. This would lead to a partial stabilization of the output of the combinational logic DAG by an argument analogous to the proof of Lemma 4.2. As permitting the non-deterministic resolution of metastability *during* the evaluation of the combinational logic is equivalent to resolving it *afterwards,* we do the latter; this is captured in the write phase defined in Section 4.5. This simplifies the analysis while being computationally equivalent to allowing partial stabilization during the evaluation.

In order to describe circuits that comprise only combinational logic and to also capture — possibly inconsistent — stabilization within the combinational logic, we define combinational circuits below. We analyze them in Chapter 7. In particular, we formalize and prove the claim that they are equivalent to combinational logic.

## 4.4 Circuits

With registers, gates, and combinational logic — see Sections 4.1–4.3 — at our disposal, we proceed with a formal circuit definition. We specify how it behaves in Section 4.5 and give an example in Section 4.6.

**Definition 4.3** (Circuit). *A* circuit *$C$ is defined by:*

*(1)* $m$ input registers, $k$ local registers, *and* $n$ output registers, *where* $m, k, n \in \mathbb{N}_0$. *Each register has exactly one* type *— simple, mask-*0*, or mask-*1 *(see Section 4.1) — and is either input, output, or local register.*

*(2) A combinational logic DAG $G$ as defined in Section 4.3. $G$ has $m + k$ input nodes, exactly one for each non-output register, and $k + n$ output nodes, exactly one for each non-input register. Local registers appear as both input node and output node.*

*(3) An initialization $x_0 \in \mathbb{B}_{\mathrm{M}}^{k+n}$ of the non-input registers.*

*Each $s \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$ defines a* state *of $C$. We refer to $C$ as* simple *if it only uses simple registers and as* combinational *if it has no local registers.*

A meaningful application clearly uses a stable initialization $x_0 \in \mathbb{B}^{k+n}$; this restriction, however, is not formally required. Furthermore, the initialization of the output registers is irrelevant because output registers are never read, see Section 4.5.

Combinational circuits are circuits that do not carry a state from one round into the next. In circuit design, a combinational circuit comprises only combinational logic, but no registers; our definition, however, only requires the absence of local registers. This removes the feedback loop from Figure 4.1(a). Carefully note that the required input and output registers are just an artifact of our definition, the "registers" of a combinational circuit can be interpreted as input and output pins. We analyze combinational circuits and show that they are equivalent to our definition of combinational logic in Chapter 7.

Observe that Definition 4.3 does not allow registers to be input and output registers at the same time, an overlap in responsibilities that is often used in digital circuits. Note, however, that we impose this restriction for purely technical reasons; our model supports registers that are read and written — local registers — and it is possible to emulate the abovementioned behavior.[3] Hence, this formal restriction has no practical implications.

[3]In the first round, copy the input into local registers. Use the local registers in the role where they are both read and written in every round. Where needed, copy the content of the local registers to output registers in every round.

We denote by

$$\mathrm{In}\colon \mathbb{B}_{\mathrm{M}}^{m+k+n} \to \mathbb{B}_{\mathrm{M}}^{m}, \tag{4.10}$$

$$\mathrm{Loc}\colon \mathbb{B}_{\mathrm{M}}^{m+k+n} \to \mathbb{B}_{\mathrm{M}}^{k}, \text{ and} \tag{4.11}$$

$$\mathrm{Out}\colon \mathbb{B}_{\mathrm{M}}^{m+k+n} \to \mathbb{B}_{\mathrm{M}}^{n} \tag{4.12}$$

the projections of circuit state to its values at input, local, and output registers, respectively. We use the convention that for any circuit state $s$, $s = \mathrm{In}(s) \circ \mathrm{Loc}(s) \circ \mathrm{Out}(s)$.

## 4.5 Executions

Consider a circuit $C$ in state $s$ and let $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$ be the state of the non-output registers. Suppose each register $R$ is read, i.e., makes a non-dashed state transition according to its type, state, and corresponding state machine in Figures 4.2(d)–4.2(f). This state transition yields a value read from, as well as a new state for, $R$. In general, this is a non-deterministic process as the state machines of the masking registers are non-deterministic. We denote by

$$\mathrm{Read}^{C}\colon \mathbb{B}_{\mathrm{M}}^{m+k} \to \mathcal{P}\left(\mathbb{B}_{\mathrm{M}}^{m+k}\right) \tag{4.13}$$

the function mapping $x$ to the set of all possible values read from non-output registers of $C$ depending on $x$. When only simple registers are involved, the read operation is deterministic:

**Observation 4.4.** *In a simple circuit $C$,* $\mathrm{Read}^{C}(x) = \{x\}$.

*Proof.* By Figure 4.2(d), each simple register in state $x \in \mathbb{B}_{\mathrm{M}}$ has output $x$. □

In the presence of masking registers, $x \in \mathrm{Read}^{C}(x)$ can occur, but the output may partially stabilize:

**Observation 4.5.** *Consider a circuit $C$ in state $s$. Then for* $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$

$$x \in \mathrm{Read}^{C}(x) \text{ and} \tag{4.14}$$

$$\mathrm{Read}^{C}(x) \subseteq \mathrm{Res}_{\mathrm{M}}(x). \tag{4.15}$$

*Proof.* Check the state transitions in Figures 4.2(e) and 4.2(f). For Equation (4.14), observe that for all state machines and all states $b \in \mathbb{B}_{\mathrm{M}}$ a state transition with output $b \in \mathbb{B}_{\mathrm{M}}$ starts in state $b$. Regarding (4.15), observe that registers in state M are not restricted by the claim and registers of any type in state $b \in \mathbb{B}$ are deterministically read as $b \in \mathrm{Res}_{\mathrm{M}}(b) = \{b\}$. □

Let $G$ be the combinational logic DAG of $C$ with $m+k$ input and $k+n$ output nodes. Suppose $o \in \mathbb{B}_{\mathrm{M}}^{m+k}$ is read from the non-output registers. Then the combinational logic of $C$ evaluates to $f^{G}(o)$, uniquely determined by $G$ and $o$. We denote all possible results of first reading the registers of $C$ w.r.t. $x$ and then evaluating the result by $\mathrm{Eval}^{C}(x)$:

$$\mathrm{Eval}^{C}\colon \mathbb{B}_{\mathrm{M}}^{m+k} \to \mathcal{P}\left(\mathbb{B}_{\mathrm{M}}^{k+n}\right), \tag{4.16}$$

$$\mathrm{Eval}^{C}(x) := \left\{f^{G}(o) \mid o \in \mathrm{Read}^{C}(x)\right\}. \tag{4.17}$$

When registers are written, we allow, but do not require, signals to stabilize. If the combinational logic evaluates the new values for the non-input registers to $\bar{x} \in \mathbb{B}_{\mathrm{M}}^{k+n}$, their new state is in $\mathrm{Res}_{\mathrm{M}}(\bar{x})$; the input registers are never overwritten. We denote this by

$$\mathrm{Write}^C \colon \mathbb{B}_{\mathrm{M}}^{m+k} \to \mathcal{P}\left(\mathbb{B}_{\mathrm{M}}^{k+n}\right), \tag{4.18}$$

$$\mathrm{Write}^C(x) := \bigcup_{\bar{x} \in \mathrm{Eval}^C(x)} \mathrm{Res}_{\mathrm{M}}(\bar{x}). \tag{4.19}$$

Observe that this is where metastability can cause inconsistencies: If a gate is read as M and this is copied to three registers, it is possible that one stabilizes to 0, one to 1, and one remains M. This is an important property of modeling metastability. Furthermore, recall from Section 4.3 that permitting stabilization at this point is computationally equivalent to allowing — possibly inconsistent — partial stabilization during the evaluation of the combinational logic DAG.

For the sake of presentation, we write $\mathrm{Read}^C(s)$, $\mathrm{Eval}^C(s)$, and $\mathrm{Write}^C(s)$ for a circuit state $s \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$, meaning that the irrelevant part of $s$ is ignored.

Let $s_r$ be a state of $C$. A *successor state* $s_{r+1}$ *of* $s_r$ is any state that can be obtained from $s_r$ as follows.

**Read phase** First read all registers, resulting in read values $o \in \mathrm{Read}^C(s_r)$. Denote by $\iota_{r+1} \in \mathbb{B}_{\mathrm{M}}^{m}$ the state of the input registers after the state transitions leading to reading $o$.

**Evaluation phase** Then evaluate the combinational logic according to the result of the read phase to $\bar{x}_{r+1} = f^G(o) \in \mathrm{Eval}^C(s_r)$.

**Write phase** Pick a partial resolution $x_{r+1} \in \mathrm{Res}_{\mathrm{M}}(\bar{x}_{r+1}) \subseteq \mathrm{Write}^C(s_r)$ of the result of the evaluation phase. Observe that the possible choices for $x_{r+1}$ depend on the read phase. The successor state is $s_{r+1} = \iota_{r+1} \circ x_{r+1}$.

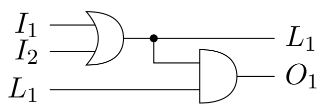
In each clock cycle, our model determines some successor state of the current state of the circuit; we refer to this as *round*.

Note that due to worst-case propagation of metastability, the evaluation phase is deterministic, while read and write phase are not: Non-determinism in the read phase is required to model the non-deterministic read behavior of masking registers, and non-determinism in the write phase allows copies of metastable bits to stabilize inconsistently. In a physical circuit, metastability may resolve within the combinational logic; as argued in Section 4.3, however, we do not model this as a non-deterministic evaluation phase, as it is equivalent to postpone possible stabilization to the write phase.

Let $C$ be a circuit in state $s_0$. For $r \in \mathbb{N}_0$, an *$r$-round execution (w.r.t. $s_0$) of* $C$ is a sequence of successor states $s_0, s_1, \ldots, s_r$. We denote by $S_r^C(s_0)$ the set of possible states resulting from $r$-round executions w.r.t. $s_0$ of $C$:

$$S_0^C(s_0) := \{s_0\} \text{ and} \tag{4.20}$$

$$S_r^C(s_0) := \left\{s_r \mid s_r \text{ is a successor state of some } s \in S_{r-1}^C(s_0)\right\}. \tag{4.21}$$



**(a)** The Circuit.

| $r$ | state $s_r$ | | | | read $o$ | | | eval $\bar{x}_{r+1}$ | | write $x_{r+1}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $I_1$ | $I_2$ | $L_1$ | $O_1$ | $I_1$ | $I_2$ | $L_1$ | $L_1$ | $O_1$ | $L_1$ | $O_1$ |
| 0 | M | M | 1 | 1 | 0 | M | 1 | M | M | 1 | M |
| 1 | M | M | 1 | M | M | M | 1 | M | M | M | M |
| 2 | 1 | M | M | M | 1 | M | M | 1 | M | 1 | 0 |
| 3 | 1 | M | 1 | 0 | 1 | M | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | M | 1 | 1 | | | | | | | |

**(b)** States, reads, evaluations, and writes.

**Figure 4.6:** Example execution in a circuit (a). The node states as well as the results of the read, evaluation, and write phases are listed in the table (b). Register $I_1$ is a mask-0 register, all others are simple registers. The initialization is 11, the input is MM, and hence $s_0 = \text{MM11}$.

The *initial state of* $C$ *w.r.t. input* $\iota \in \mathbb{B}_{\text{M}}^m$ is $s_0 = \iota \circ x_0$. We use $C_r \colon \mathbb{B}_{\text{M}}^m \to \mathcal{P}(\mathbb{B}_{\text{M}}^n)$ as a function mapping an input of $C$ to all possible outputs resulting from $r$-round executions of $C$:

$$C_r(\iota) := \left\{ \text{Out}(s_r) \mid s_r \in S_r^C(\iota \circ x_0) \right\}. \tag{4.22}$$

We say that $r$ *rounds of* $C$ *implement* $f \colon \mathbb{B}_{\text{M}}^m \to \mathcal{P}(\mathbb{B}_{\text{M}}^n)$ if and only if $C_r(\iota) \subseteq f(\iota)$ for all $\iota \in \mathbb{B}_{\text{M}}^m$, i.e., if all $r$-round executions of $C$ result in an output permitted by $f$. If there is some $r \in \mathbb{N}$, such that $r$ rounds of $C$ implement $f$, we say that $C$ *implements* $f$.

Observe that our model behaves exactly like a traditional, deterministic, binary circuit model if $s_0 \in \mathbb{B}^{m+k+n}$.

## 4.6 Example

We use this section to present an example of our model. Figure 4.6 specifies a circuit and its states, as well as the results of the read, evaluation, and write phases. The combinational logic DAG implements $f^G(x) = (x_1 \vee x_2, (x_1 \vee x_2) \wedge x_3)$. The input registers are $I_1$ and $I_2$, the only local register is $L_1$, and the only output register is $O_1$. Regarding register types, the input register $I_1$ is a mask-0 register and all other registers are simple registers.

The initialization is $x_0 = 11$, the input is $\iota = \text{MM}$, and the initial state hence is $s_0 = \iota \circ x_0 = \text{MM11}$, which is indicated in the upper left entry in Figure 4.6(b). In the read phase, all non-output registers are read. Since $I_2$ and $L_1$ are simple registers, their read deterministically evaluates to M and 1, respectively, by the state machine in Figure 4.2(d). The mask-0 register $I_1$ in state M may either be read as 0 and remain in state M, or be read as M and transition to state 1, compare Figure 4.2(e); in this case it does the former. So far, we fixed the outcome of the read phase, 0M1, and the follow-up state of the input registers, MM; the other registers are overwritten at the end of the write phase. The evaluation is uniquely determined, a read phase resulting in $o$ evaluates to $f^G(o)$, here, $f^G(0\mathrm{M}1) = \mathrm{MM}$. We are left with only one more step in this round: The non-input registers are overwritten with some value in the resolution of the evaluation phase's result, in our case with $1\mathrm{M} \in \mathrm{Res}_\mathrm{M}(\mathrm{MM})$. Together we obtain the successor state $s_1 = \mathrm{MM1M}$.

In the next round, $I_1$ follows the other state transition, i.e., is read as M, and hence has state 1 in the next round. Its state thus remains fixed in all successive rounds by the state machine in Figure 4.2(e). The other reads are deterministic, so we obtain $o = \mathrm{MM}1$ as the result of the read phase and successor states 1M for $I_1$ and $I_2$. The evaluation is $f^G(o) = f^G(\mathrm{MM}1) = \mathrm{MM}$ and the state of $L_1$ and $O_1$ is overwritten with some value from $\mathrm{Res}_\mathrm{M}(\mathrm{MM})$, here, by MM.

By round $r = 2$, the result of the read phase is deterministic because the only masking register stabilized, we read $o = 1\mathrm{MM}$, and evaluate to 1M. The remaining non-determinism is whether to write 1M or some stabilization thereof; we examine the case that 10 is written.

Rounds $r \geq 3$ now are entirely deterministic. The only possible read is 1M1, which evaluates to $f^G(1\mathrm{M}1) = 11$, fixing the result of the write phase to 11. Further rounds are identical: The only metastable register, $I_2$, remains metastable but has no impact on the evaluation phase as the Or gate always receives input 1 from $I_2$ and hence masks the metastable input.

## 4.7 Basic Properties

In this section, we establish basic properties regarding computability in the model proposed above. Regarding the implementability of functions by circuits, we focus on three resources: the number $r \in \mathbb{N}$ of rounds, the register types available to it, and whether the circuit is combinational.

**Definition 4.6** ($\mathrm{Fun}_M^r$, $\mathrm{Fun}_S^r$, and $\mathrm{Fun}_C^r$)**.** *Let* $\mathrm{Fun}_M^r$ *be the class of functions implementable with r rounds of arbitrary circuits. Analogously,* $\mathrm{Fun}_S^r$ *denotes the class of functions implementable with r rounds in simple circuits, i.e., by circuits that may only use simple registers. With* $\mathrm{Fun}_C^r$*, we refer to the class of functions implementable with r rounds of a combinational circuit, i.e., by circuits without local registers.*

Our first claim is that the result of the write phase, i.e., $\mathrm{Write}^C$, is not influenced by the register types the circuit $C$ uses. Carefully note that the successor state is influenced by the register types, as it accounts for the state transitions that registers make in the read phase. The reason that $\mathrm{Write}^C$ does not depend on the register types is as follows. By Observations 4.4 and 4.5, simple registers in state $x$ are deterministically read as $x$, and masking registers in state $x$ either as $x$ or as some $x' \in \mathrm{Res}_\mathrm{M}(x)$. Hence, the use of masking registers might partially stabilize the input to the combinational logic which, by Lemma 4.2, partially stabilizes its output. The same stabilization, however, can also occur in the write phase.

**Lemma 4.7.** *Consider a circuit $C$ in state $s$. Let $C_S$ be a copy of $C$ that only uses simple registers and $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$ the projection of $s$ to the non-output registers. Then*

$$\mathrm{Write}^C(s) = \mathrm{Write}^{C_S}(s) = \mathrm{Res}_{\mathrm{M}}\left(f^G(x)\right). \tag{4.23}$$

*Proof.* In $C_S$, we have $\mathrm{Read}^{C_S}(s) = \{x\}$ by Observation 4.4. So $\mathrm{Eval}^{C_S}(s) = \{f^G(x)\}$, and $\mathrm{Write}^{C_S}(s) = \mathrm{Res}_{\mathrm{M}}(f^G(x))$ by definition.

In $C$, $x \in \mathrm{Read}^C(s)$ by Observation 4.5, so $\mathrm{Res}_{\mathrm{M}}(f^G(x)) \subseteq \mathrm{Write}^C(s)$. All other reads $x' \in \mathrm{Read}^C(s)$ have $x' \in \mathrm{Res}_{\mathrm{M}}(x)$ by Observation 4.5 and $f^G(x') \in \mathrm{Res}_{\mathrm{M}}(f^G(x))$ by Lemma 4.2. It follows that $\mathrm{Write}^C(s) = \mathrm{Res}_{\mathrm{M}}(f^G(x))$. □

Carefully note that the write phase only affects non-input registers; input registers are never written. Hence, Lemma 4.7 does not generalize to multiple rounds: State transitions of input registers in the read phase affect future read phases.

In 1-round executions, however, masking and simple registers are equally powerful, because their state transitions only affect rounds $r \geq 2$. We show in Chapter 8 that these state changes may indeed lead to differences for $r \geq 2$ rounds if $C$ contains masking registers.

**Corollary 4.8.** $\mathrm{Fun}^1_C = \mathrm{Fun}^1_S = \mathrm{Fun}^1_M$.

In contrast, simple and masking registers used as non-input registers behave identically, regardless of the number of rounds: A circuit $C$ in state $s_r$ overwrites them regardless of their state. Since $\mathrm{Write}^C(s_r)$ is oblivious to register types by Lemma 4.7, so is $\mathrm{Loc}(s_{r+1}) \circ \mathrm{Out}(s_{r+1})$ for a successor state $s_{r+1}$ of $s_r$.

**Corollary 4.9.** *Simple and masking registers are interchangeable when used as non-input registers.*

Consider a circuit $C$ in state $s$ and suppose $x \in \mathrm{Read}^C(s)$ is read. Since the evaluation phase is deterministic, the evaluation $y = f^G(x) \in \mathrm{Eval}^C(s)$ is uniquely determined by $x$ and $C$. Recall that we may resolve metastability to $\mathrm{Res}_{\mathrm{M}}(y) \subseteq \mathrm{Write}^C(s)$ in the write phase: The state of an output register $R$ becomes 0 if $y_R = 0$, 1 if $y_R = 1$, and some $b \in \mathbb{B}_{\mathrm{M}}$ if $y_R = \mathrm{M}$. Consequently, output registers resolve independently:

**Corollary 4.10.** *Let $C$ be a circuit. Then the output bits of $C$ after one round resolve independently, i.e.,*

$$C_1 = g_0 \times \cdots \times g_{n-1}, \tag{4.24}$$

*where $g_i \colon \mathbb{B}^m_{\mathrm{M}} \to \{\{0\}, \{1\}, \mathbb{B}_{\mathrm{M}}\}$.*

*Proof.* Let $s = \iota \circ x_0$ be the initial state of $C$ w.r.t. input $\iota$ and $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$. By Lemma 4.7, $\mathrm{Write}^C(s) = \mathrm{Res}_{\mathrm{M}}(f^G(x))$, i.e., $C_1(\iota) = \{\mathrm{Out}(s') \mid s' \in \mathrm{Res}_{\mathrm{M}}(f^G(x))\}$. By definition, $\mathrm{Res}_{\mathrm{M}}(f^G(x)) = \prod_{i \in [n]} \mathrm{Res}_{\mathrm{M}}(f^G(x))_i$. Hence, the claim follows with $g_i(\iota) := \mathrm{Res}_{\mathrm{M}}(f^G(\iota \circ \mathrm{Loc}(x_0)))_i$ for all $\iota \in \mathbb{B}^m_{\mathrm{M}}$ and $i \in [n]$. □

We show in Chapter 9 that Corollary 4.10 generalizes to multiple rounds of simple circuits. This is, however, not the case in the presence of masking registers, as demonstrated in Chapter 8.

Lemmas 4.2 and 4.7 apply to the input of circuits: Partially stabilizing an input partially stabilizes the possible inputs of the combinational logic, and hence its evaluation and the circuit's output after one round.

**Observation 4.11.** *For a circuit $C$ and input $\iota \in \mathbb{B}_{\mathrm{M}}^m$,*

$$\iota' \in \mathrm{Res}_{\mathrm{M}}(\iota) \Rightarrow C_1(\iota') \subseteq C_1(\iota). \tag{4.25}$$

*Proof.* Let $x_0$ be the initialization of $C$, $s = \iota \circ x_0$ its initial state w.r.t. input $\iota$, and $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$ the state of the non-output registers; define $s'$ and $x'$ equivalently w.r.t. input $\iota' \in \mathrm{Res}_{\mathrm{M}}(\iota)$. Using Lemmas 4.2 and 4.7 and that $\mathrm{Res}_{\mathrm{M}}(x') \subseteq \mathrm{Res}_{\mathrm{M}}(x)$ for $x' \in \mathrm{Res}_{\mathrm{M}}(x)$, we obtain that

$$\mathrm{Write}^C(s') \overset{(4.23)}{=} \mathrm{Res}_{\mathrm{M}}(f^G(x')) \overset{(4.9)}{\subseteq} \mathrm{Res}_{\mathrm{M}}(f^G(x)) \overset{(4.23)}{=} \mathrm{Write}^C(s). \tag{4.26}$$

□

Finally, note that adding rounds of computation to non-combinational circuits cannot decrease computational power; a circuit determining $x$ in $r$ rounds can be transformed into one using $r+1$ rounds by buffering $x$ for one round. Furthermore, allowing masking registers does not decrease computational power and neither does allowing local registers.

**Observation 4.12.** *For all $r \in \mathbb{N}_0$ we have*

$$\mathrm{Fun}_S^r \subseteq \mathrm{Fun}_S^{r+1}, \tag{4.27}$$
$$\mathrm{Fun}_M^r \subseteq \mathrm{Fun}_M^{r+1}, \tag{4.28}$$
$$\mathrm{Fun}_S^r \subseteq \mathrm{Fun}_M^r\,, \text{ and} \tag{4.29}$$
$$\mathrm{Fun}_C^r \subseteq \mathrm{Fun}_M^r\,. \tag{4.30}$$

## 4.8 The Next Steps

The subtlety of metastability in digital circuits makes it challenging to properly model it. Especially a model that is both time-discrete and value-discrete, as the one proposed above, runs the risk of oversimplifying matters. Hence, we are obliged to address two issues.

(1) One is the question whether the proposed model is consistent with physical circuit properties: Is the model "too powerful" in the sense that it permits things that are known to be impossible in physical circuits? We address this by showing in Chapter 5 that Marino's impossibility result [108] — no digital circuit can reliably avoid, detect, or resolve metastability — holds in our model.

(2) The other question is whether the model is not "too pessimistic" and permits non-trivial positive results. We demonstrate that this is not a problem in Chapters 6 and 10, where we develop CMUXes, a core component of metastability-containing circuits, and show that all components needed for metastability-tolerant clock synchronization in hardware are within reach.

Together with further positive results developed using this model [26, 62, 100, 128], including simulations [26, 60, 128], this strongly indicates that the proposed model is both useful and makes meaningful predictions.

This document is arranged such that Chapter 5, which justifies the model, is ordered before Chapter 6, which uses it. We refer to the first sections of Chapter 6 for relevant examples on how to use the model.

# CHAPTER 5

# Justification

The elusive nature of metastability and Marino's impossibility result [108] raise two concerns about the model proposed in Chapter 4. Is is "too optimistic" to derive meaningful statements about the physical world? Or is it "too pessimistic" in the sense that it does not permit non-trivial positive results? In this chapter, we address the first concern by showing that our model reproduces Marino's impossibility result. We answer the second question in Chapter 6, where we develop CMUXes, starting with the theory at the gate-level and advancing to optimized transistor-level implementations.

Marino established that no digital circuit can reliably (1) avoid, (2) resolve, or (3) detect metastability [108]. It is critical that these impossibilities are reflected by any model comprising metastability. We show in Theorem 5.4 and Corollaries 5.5–5.6 that (1)–(3) are impossible in the model proposed in Chapter 4 as well. Note that this is about putting the model to the test rather than reproducing an established result.

We first verify that avoiding metastability is impossible in non-trivial[1] circuits. Consider a circuit $C$ that produces different outputs for inputs $\iota \neq \iota'$. The idea is to observe how the output of $C$ behaves while transforming $\iota$ to $\iota'$ bit by bit, always involving intermediate metastability, i.e., switching the differing bits from 0 to M to 1 or vice versa. This can be seen as a discrete version of Marino's argument for signals that map continuous time to continuous voltage [108]. Furthermore, the bit-wise transformation of $\iota$ to $\iota'$, enforcing a change in the output in between, has parallels to the classical impossibility of consensus proof of Fischer et al. [52]; our techniques, however, are quite different. The following definition formalizes the step-wise manipulation of bits.

**Definition 5.1** (Pivotal Sequence)**.** *Let $d \in \mathbb{N}_0$ and $\ell \in \mathbb{N}$ be integers and $x, x' \in \mathbb{B}_{\mathrm{M}}^d$. Then*

$$\left(x^{(i)}\right)_{i \in [\ell+1]}, \quad x^{(i)} \in \mathbb{B}_{\mathrm{M}}^d, \tag{5.1}$$

*is a* pivotal sequence from $x$ to $x'$ over $\mathbb{B}_{\mathrm{M}}^d$ *if and only if it satisfies*

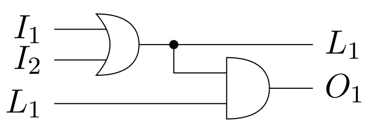
*(1) $x^{(0)} = x$ and $x^{(\ell)} = x'$,*

*(2) for all $i \in [\ell]$, $x^{(i)}$ and $x^{(i+1)}$ differ in exactly one bit, and*

*(3) that bit is metastable in either $x^{(i)}$ or $x^{(i+1)}$.*

*Consider $i \in [\ell]$. We call the differing bit in $x^{(i)}$ and $x^{(i+1)}$ the* pivot from $i$ to $i+1$. *In the context of a pivotal sequence of circuit states, we refer to the register holding the pivot as* pivotal register.

[1]We call the circuit $C$ non-trivial if there are inputs $\iota \neq \iota'$ and a number of rounds $r \in \mathbb{N}$ such that $C_r(\iota) \cap C_r(\iota') = \emptyset$, i.e., if $r$ rounds of $C$ have to produce different outputs for $\iota$ and $\iota'$.



**(a)** The Circuit.

| | $x^{(i)}$ | | | | $f^G\left(\bar{x}^{(i)}\right)$ | | $y^{(j)}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $I_1$ | $I_2$ | $L_1$ | $O_1$ | $L_1$ | $O_1$ | $I_1$ | $I_2$ | $L_1$ | $O_1$ | |
| $x^{(0)}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $y^{(0)}$ |
| $x^{(1)}$ | 0 | 0 | M | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $x^{(2)}$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $x^{(3)}$ | 0 | M | 1 | 0 | M | M | 0 | M | 0 | 0 | $y^{(1)}$ |
| | 0 | M | 1 | 0 | M | M | 0 | M | M | 0 | $y^{(2)}$ |
| | 0 | M | 1 | 0 | M | M | 0 | M | M | M | $y^{(3)}$ |
| | 0 | M | 1 | 0 | M | M | 0 | M | M | 1 | $y^{(4)}$ |
| | 0 | M | 1 | 0 | M | M | 0 | M | 1 | 1 | $y^{(5)}$ |
| $x^{(4)}$ | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | $y^{(6)}$ |

**(b)** Pivotal sequences of circuit states for $\bar{x}^{(i)} := \mathrm{In}\left(x^{(i)}\right) \circ \mathrm{Loc}\left(x^{(i)}\right)$.

**Figure 5.1:** A circuit with input ($I_1$ and $I_2$), local ($L_1$), and output ($O_1$) registers (a), all simple, and a pivotal sequence of circuit states $x^{(0)}, \ldots, x^{(4)}$ with the constructed pivotal sequence of successor states $y^{(0)}, \ldots, y^{(6)}$ (b). Each step in $x$ is reflected in a re-evaluation of the combinational logic $f^G(x^{(i)})$, which may affect several registers of the successor state. To be pivotal, the sequence $y$ accounts for these changes bit by bit.

Carefully note that we do not use pivotal sequences as temporal sequences of successor states. The bit-wise manipulation does not happen over time, instead, we describe closely related circuit states and examine the resulting 1-round executions. The generalization to $r$-round executions is made in Corollary 5.3.

We begin with Lemma 5.2, which applies to a single round of computation. It states that initializing a circuit $C$ with a pivotal sequence $x$ of circuit states results in a pivotal sequence of possible successor states $y$. Hence, if $C$ is non-trivial — $C$ guarantees different outputs for $x^{(0)}$ and $x^{(\ell)}$ — some intermediate element of $y$ must contain a metastable output bit. This implies that there is a 1-round execution in which an output register of $C$ becomes metastable. We argue about successor states rather than just the output because we inductively apply Lemma 5.2 below to obtain a statement about $r$-round executions.

A sample circuit with pivotal sequences is depicted in Figure 5.1. Observe that in this example, the local register $L_1$ changes its state. This does not occur in the first round, because the initialization for local registers is fixed. It may, however, happen in successive rounds.

Let $x$ be a pivotal sequence of non-output register states, i.e., over $\mathbb{B}_{\mathrm{M}}^{m+k}$, and suppose the pivotal register changes from stable to M from $x^{(i)}$ to $x^{(i+1)}$. By Observation 4.5, we may construct executions where $x^{(i)} \in \mathrm{Read}^C(x^{(i)})$ and $x^{(i+1)} \in \mathrm{Read}^C(x^{(i+1)})$ are the result of the respective read phases. The key insight is that due to $x^{(i)} \in \mathrm{Res}_{\mathrm{M}}(x^{(i+1)})$, we have

$$\mathrm{Write}^C\left(x^{(i)}\right) \subseteq \mathrm{Write}^C\left(x^{(i+1)}\right) \tag{5.2}$$

because

$$\begin{aligned} \mathrm{Write}^C\left(x^{(i)}\right) &\overset{(4.23)}{=} \mathrm{Res}_{\mathrm{M}}\left(f^G\left(x^{(i)}\right)\right) &(5.3)\\ &\overset{(4.9)}{\subseteq} \mathrm{Res}_{\mathrm{M}}\left(f^G\left(x^{(i+1)}\right)\right) &(5.4)\\ &\overset{(4.23)}{=} \mathrm{Write}^C\left(x^{(i+1)}\right) &(5.5)\end{aligned}$$

by Lemmas 4.2 and 4.7. Thus, advancing the non-output bits of the initial state from $x^{(i)}$ to $x^{(i+1)}$ destabilizes the bits that are affected by the pivot from $i$ to $i+1$; we leave all other bits unchanged. Leveraging this, we obtain a pivotal sequence of successor states, changing the affected output bits from stable to M one by one, each the result of a 1-round execution of $C$ starting in state $x^{(i+1)}$. A reversed version of this argument applies when a non-output register changes from M to stable.

Observe that this process does not guarantee that $x$ and $y$ have the same length; some steps in $x$ may not lead to a new successor state at all while others may require several steps for $y$ to "catch up." Figure 5.1 contains examples for both cases.

**Lemma 5.2.** *Let $C$ be a circuit and*

$$\left(x^{(i)}\right)_{i\in[\ell+1]}, \quad x^{(i)} \in \mathbb{B}_{\mathrm{M}}^{m+k+n}, \tag{5.6}$$

*a pivotal sequence of states of $C$. Then there is a pivotal sequence*

$$\left(y^{(j)}\right)_{j\in[\ell'+1]}, \quad y^{(j)} \in \mathbb{B}_{\mathrm{M}}^{m+k+n}, \tag{5.7}$$

*where each $y^{(j)}$ is a successor state of some $x^{(i)}$, satisfying that $y^{(0)}$ and $y^{(\ell')}$ are successor states of $x^{(0)}$ and $x^{(\ell)}$, respectively.*

*Proof.* See Figure 5.1 for an illustration of our arguments. Let $G$ be the combinational logic DAG of $C$. Starting from $x^{(0)}$, we inductively proceed to $x^{(\ell)}$, extending the sequence $y$ by a suitable, possibly empty, subsequence for each step from $x^{(i)}$ to $x^{(i+1)}$, $i \in [\ell]$.

We maintain the invariants that

(1) For all $i \in [\ell+1]$, $\mathrm{In}(x^{(i)}) \circ \mathrm{Loc}(x^{(i)}) \in \mathrm{Read}^C(x^{(i)})$ is the result of the read phase in all constructed executions starting in state $x^{(i)}$ and that

(2) For all $i \in [\ell+1]$, there is a $j \in [\ell'+1]$ with

$$\mathrm{Loc}\left(y^{(j)}\right) \circ \mathrm{Out}\left(y^{(j)}\right) = f^G\left(\mathrm{In}\left(x^{(i)}\right) \circ \mathrm{Loc}\left(x^{(i)}\right)\right), \tag{5.8}$$

i.e., where no stabilization happens in the write phase of the execution resulting in state $y^{(j)}$. In this context, $y^{(j)}$ is the state *corresponding to $x^{(i)}$*. This is the case for the state pairs $x^{(0)}$ and $y^{(0)}$, $x^{(1)}$ and $y^{(0)}$, $x^{(2)}$ and $y^{(0)}$, $x^{(3)}$ and $y^{(3)}$, and $x^{(4)}$ and $y^{(6)}$ in Figure 5.1.

Let $\iota := \mathrm{In}(x^{(0)})$ be the state of the input registers. Lemma 4.7 guarantees that we have $f^G(\iota \circ \mathrm{Loc}(x^{(0)})) \in \mathrm{Write}^C(x^{(0)})$. Define

$$y^{(0)} := \iota' \circ f^G\left(\iota \circ \mathrm{Loc}\left(x^{(0)}\right)\right) \in S_1^C\left(x^{(0)}\right), \tag{5.9}$$

where $\iota'$ is the uniquely determined state of the input registers after reading $\iota$. By construction, $x^{(0)}$ and $y^{(0)}$ fulfill the invariants.

We perform the step from $x^{(i)}$ to $x^{(i+1)}$, $i \in [\ell]$. Let $P$ be the pivotal register from $x^{(i)}$ to $x^{(i+1)}$. If $P$ is an output register, $y$ does not change as $P$ has no impact on the successor state, trivially completing the induction step. Hence, assume that $P$ is an input or local register. From the previous step, or from the definition of $y^{(0)}$, we have an execution resulting in state $y^{(j)}$ for some index $j$, such that $\mathrm{In}(x^{(i)}) \circ \mathrm{Loc}(x^{(i)})$ is the result of the read phase and $y^{(j)}$ corresponds to $x^{(i)}$. For the next step, we keep the result of the read phase for all registers except $P$ fixed. Regarding all registers that do not depend on $P$, i.e., may attain the same states regardless of what is read from $P$, we rule that they attain the same states as in $y^{(j)}$, the state corresponding to $x^{(i)}$.

Suppose first that $x_P^{(i)} = b \neq \mathrm{M}$ and $x_P^{(i+1)} = \mathrm{M}$ (e.g., the step from $x^{(2)}$ to $x^{(3)}$ with $P = I_2$ in Figure 5.1). Consider the set of non-input registers $\mathcal{R}$ that depend on $P$, i.e.,

$$\mathcal{R} := \left\{R \mid f^G\left(x^{(i)}\right)_R \neq f^G\left(x^{(i+1)}\right)_R\right\} \tag{5.10}$$

($\mathcal{R} = \{L_1, O_1\}$ in the step from $x^{(2)}$ to $x^{(3)}$ in Figure 5.1). Since $x^{(i)} \in \mathrm{Res}_{\mathrm{M}}(x^{(i+1)})$, by Lemma 4.2 $f^G(x^{(i)}) \in \mathrm{Res}_{\mathrm{M}}(f^G(x^{(i+1)}))$. Hence,

$$f^G\left(x^{(i+1)}\right)_R = \mathrm{M} \neq f^G\left(x^{(i)}\right)_R \tag{5.11}$$

for all $R \in \mathcal{R}$.

If $P$ is an input register, we first extend $y$ by one item that only changes $y_P$ to M and increase $j$ by one if that is the case (e.g., the step from $y^{(0)}$ to $y^{(1)}$ in Figure 5.1). Then we extend $y$ by $y^{(j+1)}, \ldots, y^{(j+|\mathcal{R}|)}$ such that in each step, for one $R \in \mathcal{R}$, we change $y_R$ from $b_R \neq \mathrm{M}$ to M. This is feasible by Corollary 4.10, as the product structure of $C_1$ implies that we can flip any written bit without affecting the others (e.g., steps $y^{(2)}$ and $y^{(3)}$ in Figure 5.1). By construction, in state $y^{(j+|\mathcal{R}|)}$ the state of the non-input registers is $f^G(\iota \circ x^{(i+1)})$, i.e., $y^{(j+|\mathcal{R}|)}$ corresponds to $x^{(i+1)}$ and our invariant is satisfied.

To cover the case that $x_P^{(i)} = \mathrm{M}$ and $x_P^{(i+1)} = b \neq \mathrm{M}$, observe that we can apply the same reasoning by reversing the order of the constructed subsequence.

As $y$ is pivotal by construction — we change only one bit at a time and always switch from M to stable or vice versa — this completes the proof. □

We may apply Lemma 5.2 inductively: Starting with a pivotal sequence of initial states, we obtain executions leading to a pivotal sequence of circuit states after one round, these in turn lead to two-round executions and a pivotal sequence of circuit states after two rounds, and so on.

**Corollary 5.3.** *Let $C$ be a circuit, $x_0$ its initialization, and*

$$\left(\iota^{(i)}\right)_{i\in[\ell+1]}, \quad \iota^{(i)} \in \mathbb{B}_{\mathrm{M}}^m, \tag{5.12}$$

*be a pivotal sequence of inputs of $C$. Then there is a pivotal sequence of states*

$$\left(y^{(j)}\right)_{j\in[\ell'+1]}, \quad y^{(j)} \in \mathbb{B}_{\mathrm{M}}^{m+k+n}, \tag{5.13}$$

*that $C$ can attain after $r \in \mathbb{N}$ rounds satisfying that $y^{(0)} \in S_r^C(\iota^{(0)} \circ x_0)$ as well as $y^{(\ell')} \in S_r^C(\iota^{(\ell)} \circ x_0)$.*

*Proof.* Inductive application of Lemma 5.2 to $C$ and states $\left(\iota^{(i)} \circ x_0\right)_{i\in[\ell+1]}$. □

The following theorem concludes that any non-trivial circuit can produce metastable outputs, i.e., that the first of Marino's impossibility results applies to the model proposed in Chapter 4.

**Theorem 5.4.** *Let $C$ be a circuit with $C_r(\iota) \cap C_r(\iota') = \emptyset$ for some $\iota, \iota' \in \mathbb{B}_{\mathrm{M}}^m$. Then $C$ has an $r$-round execution in which an output register becomes metastable.*

*Proof.* Apply Corollary 5.3 to a pivotal sequence from $\iota$ to $\iota'$ and $C$. This yields a pivotal sequence $y$ of states that $C$ can attain after $r$-round executions. Consider

$$\left(\bar{y}^{(i)}\right)_{i\in[\ell+1]}, \quad \bar{y}^{(i)} \in \mathbb{B}_{\mathrm{M}}^{n}, \tag{5.14}$$

the projection of $y$ to the output registers that skips succeeding, identical elements. Since $C_r(\iota) \cap C_r(\iota') = \emptyset$, $\bar{y}$ is a pivotal sequence that contains at least two elements. Hence, $\bar{y}_R^{(i)} = \mathrm{M}$ for some output register $R$ and some $i \in [\ell+1]$, i.e., there is an $r$-round execution in which $C$ outputs a metastable bit, as claimed. □

Marino proved that no digital circuit can reliably (1) compute a non-constant function and guarantee non-metastable output, (2) detect whether a register is metastable, or (3) resolve metastability of the input while faithfully propagating stable input [108]. Theorem 5.4 captures (1); Corollaries 5.5 and 5.6 settle (2) and (3), respectively. The key is that a circuit detecting or resolving metastability is non-constant and can hence become metastable by Theorem 5.4, defeating the purpose of detecting or resolving metastability in the first place.

**Corollary 5.5.** *There exists no circuit that implements $f\colon \mathbb{B}_{\mathrm{M}} \to \mathcal{P}(\mathbb{B}_{\mathrm{M}})$ with*

$$f(x) = \begin{cases} \{1\} & \text{if } x = \mathrm{M} \text{ and} \\ \{0\} & \text{otherwise.} \end{cases} \tag{5.15}$$

*Proof.* Assume such a circuit $C$ exists and implements $f$ using $r$ rounds. As we have $C_r(0) \cap C_r(\mathrm{M}) = \emptyset$, applying Theorem 5.4 to $\iota = 0$ and $\iota' = \mathrm{M}$ yields that $C$ has an $r$-round execution with metastable output. This violates the specification in Equation (5.15), contradicting the assumption. □

**Corollary 5.6.** *There exists no circuit that implements $f\colon \mathbb{B}_{\mathrm{M}} \to \mathcal{P}(\mathbb{B}_{\mathrm{M}})$ with*

$$f(x) = \begin{cases} \{0,1\} & \text{if } x = \mathrm{M} \text{ and} \\ \{x\} & \text{otherwise.} \end{cases} \tag{5.16}$$

*Proof.* As in Corollary 5.5 with $\iota = 0$ and $\iota' = 1$. $\square$

In summary, our circuit model from Chapter 4 is consistent with physical models of metastability. We demonstrate in Chapters 6 and 10 that it also permits non-trivial positive results and further examine its computational power in Chapters 7–9.

# CHAPTER 6

# Metastability-Containing Multiplexers

In this chapter, we develop Metastability-Containing Multiplexers (CMUXes). The basis of our discussion are gate-level implementations, which are a good example on how to use our model because (1) they have the right level of complexity and (2) metastability-containment makes a difference in MUXes in the sense that a MUX and a CMUX behave differently. From a broader perspective, this demonstrates that our model, especially the worst-case propagation of metastability, is not "too pessimistic" to permit positive results while Chapter 5 shows that it is not "too optimistic," either.

As MUXes are ubiquitous in digital circuits, we also study efficient transistor-level CMUX implementations. The idea is that a small CMUX implementation can be used as drop-in replacement to improve existing circuits. Bund et al. ask whether such an implementation exists [26], as existing metastability-containing sorting networks heavily depend on CMUXes [26, 100]. Our smallest CMUX implementation uses only 8 transistors — no more than a standard[1] transistor-level MUX — and dramatically decreases the size of the metastability-containing sorting networks of Lenzen and Medina [100] and of Bund et al. [26], saving up to 69 % and 65 % in circuit complexity, respectively.

In this chapter, we use the following convention to simplify the specification of small combinational circuits.

**Remark 6.1** (Notation). *We specify some combinational circuits, e.g.,* MUX-GATE *in Figure 6.1(a), only by their combinational logic DAG. The circuit in terms of Definitions 4.3 which we mean by that is the one that (1) only uses simple input and arbitrarily initialized output registers,[2] (2) has no local registers, and (3) uses the given combinational logic DAG. We only follow this convention for combinational circuits which are equivalent to the combinational logic and w.l.o.g. use a single round of computation by Theorem 7.1.*

## 6.1 Problem Statement

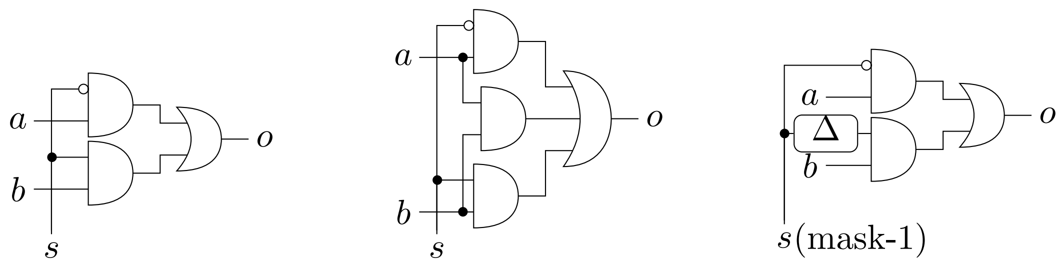
Prior to discussing metastability-containing variants, let us examine a standard MUX. An *(n-bit) Multiplexer (MUX)* is a circuit with $2n + 1$ inputs that implements

$$f_{\mathrm{MUX}} \colon \mathbb{B}_{\mathrm{M}} \times \mathbb{B}_{\mathrm{M}}^n \times \mathbb{B}_{\mathrm{M}}^n \to \mathbb{B}_{\mathrm{M}}^n \tag{6.1}$$

$$f_{\mathrm{MUX}}(s,a,b) := \begin{cases} \mathrm{Res}_{\mathrm{M}}(a) & \text{if } s = 0, \\ \mathrm{Res}_{\mathrm{M}}(b) & \text{if } s = 1, \text{ and} \\ \mathbb{B}_{\mathrm{M}}^n & \text{otherwise.} \end{cases} \tag{6.2}$$

[1] Pass-gate MUXes need 4 transistors but are inherently vulnerable to metastability, see Section 6.3.2.

[2] The initialization is irrelevant as the output registers do not influence the computation.



**(a)** Mux-Gate. **(b)** CMux-Gate. **(c)** CMux-Delay.

**Figure 6.1:** Gate-level MUX implementations. Mux-Gate (a) is a standard gate-level MUX and can become metastable, see Figure 6.2. CMux-Gate (b) and CMux-Delay (c) avoid this by employing additional gates and a masking register, respectively.

For the sake of presentation, we use $n = 1$ throughout the chapter; all presented circuits generalize to higher values of $n$ by replication. We refer to $s$ as the *select bit.* In the case of a stable select bit, a MUX determines whether to output (some stabilization of) $a$ or $b$. If $s$ is metastable, the output is unspecified. Figure 6.1(a) shows Mux-Gate, a typical gate-level implementation of a MUX.

Note that a real-world circuit typically uses an efficient transistor-level MUX instead of a gate-level implementation like Mux-Gate. The gate level, however, is captured by our model from Chapter 4. Furthermore, it is well-suited for demonstrating the basic problem with standard MUXes that is also present — although harder to see — in transistor-level MUXes. We discuss this in Section 6.3.

A desirable property of a MUX is that if $a = b$, the output is $\mathrm{Res}_{\mathrm{M}}(a)$, regardless of $s$: Being uncertain whether to select $a$ or $b$ should be insubstantial in this case. If, however, $s = \mathrm{M}$ and $a = b = 1$, a standard implementation like Mux-Gate can become metastable, compare Figure 6.2(a).

**Observation 6.2.** *Let $C^{\textsc{Mux-Gate}}$ be the circuit associated with* Mux-Gate *in the sense of Remark 6.1. As depicted in Figure 6.2(a), $C^{\textsc{Mux-Gate}}$ can become metastable when provided with input $s = \mathrm{M}$ and $a = b = 1$, i.e., $C_1^{\textsc{Mux-Gate}}(\mathrm{M}, 1, 1) = \mathbb{B}_{\mathrm{M}}$. This follows from observing that $(s, a, b)$ can be read as $(\mathrm{M}, 1, 1)$ by Observation 4.5, evaluating the combinational logic DAG accordingly*

$$(\neg s \wedge a) \vee (s \wedge b) = (\neg \mathrm{M} \wedge 1) \vee (\mathrm{M} \wedge 1) = \mathrm{M} \vee \mathrm{M} = \mathrm{M}, \tag{6.3}$$

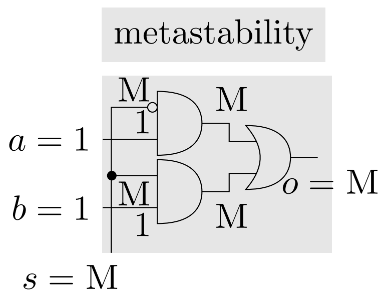
*and recalling that this implies* $\mathrm{Write}^{C^{\textsc{Mux-Gate}}}(\mathrm{M}, 1, 1) = \mathrm{Res}_{\mathrm{M}}(\mathrm{M}) = \mathbb{B}_{\mathrm{M}}$.

Hence, we ask for an *(n-bit) Metastability-Containing Multiplexer (CMUX),* an improved circuit that implements

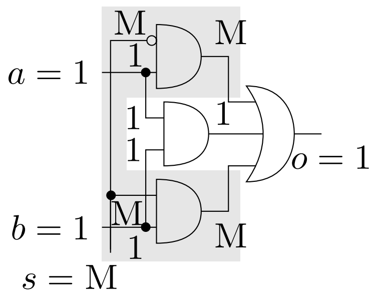
$$f_{\mathrm{CMUX}} \colon \mathbb{B}_{\mathrm{M}} \times \mathbb{B}_{\mathrm{M}}^n \times \mathbb{B}_{\mathrm{M}}^n \to \mathbb{B}_{\mathrm{M}}^n \tag{6.4}$$

$$f_{\mathrm{CMUX}}(s, a, b) := \begin{cases} \mathrm{Res}_{\mathrm{M}}(a) & \text{if } s = 0, \\ \mathrm{Res}_{\mathrm{M}}(b) & \text{if } s = 1, \text{ and} \\ \mathrm{Res}_{\mathrm{M}}(a \oplus b) & \text{otherwise,} \end{cases} \tag{6.5}$$

i.e., which has to output $\mathrm{Res}_{\mathrm{M}}(a)$ if $a = b$, even if $s = \mathrm{M}$.



**(a)** Mux-Gate under input $s = \mathrm{M}$ and $a = b = 1$.



**(b)** CMux-Gate under input $s = \mathrm{M}$ and $a = b = 1$.

**Figure 6.2:** Gate-level MUX behavior under inputs $(s, a, b) = (\mathrm{M}, 1, 1)$; in this case, the output should be 1 regardless of the select bit $s$. Mux-Gate (a), however, can become metastable. CMux-Gate (b) fixes this using logical masking.

## 6.2 Gate-Level Implementations

The circuit[3] CMux-Gate in Figure 6.1(b) implements $f_{\mathrm{CMUX}}$ from Equation (6.5): The problematic case of $s = \mathrm{M}$ and $a = b = 1$ is handled by the third And gate which outputs 1, providing the ternary Or gate with a stable 1 as input, compare Figure 6.2(b).

**Lemma 6.3.** *One round of $C^{\text{CMux-Gate}}$ implements $f_{CMUX}$ from Equation* (6.5)*.*

*Proof.* $C^{\text{CMux-Gate}}$ has no local registers and its combinational logic DAG implements

$$o = (\neg s \wedge a) \vee (s \wedge b) \vee (a \wedge b). \tag{6.6}$$

It is easy to verify that this implements $f_{\mathrm{CMUX}}$ from Equation (6.5) for $s \neq \mathrm{M}$. Hence, consider $s = \mathrm{M}$ in the following. If $a \neq b$ or $a = b = \mathrm{M}$, the output of $C^{\text{CMux-Gate}}$ is not restricted by Equation (6.5), so consider the remaining two cases:

**Case 1** In the case of $a = b = 0$, all clauses in Equation (6.6) are 0, leading to $o = 0$.

**Case 2** If $a = b = 1$, we have $a \wedge b = 1$ and $o = 1$, regardless of the other clauses.

Together, this concludes the proof. □

The price for this improvement is an additional And gate and a ternary instead of a binary Or gate, which can be costly if $a$ and $b$ have a large bit width $n$. Lenzen and Medina allocate 28 transistors to implement CMux-Gate, 2 for the Not, 6 for each And, and 8 for the ternary Or [100]. This can be improved to 20 using Nand gates and the de Morgan equivalent of Equation (6.6),

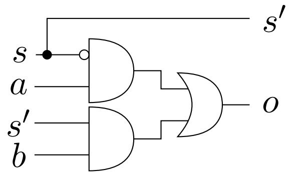
$$(\neg s \wedge a) \vee (s \wedge b) \vee (a \wedge b) = \neg(\neg(\neg s \wedge a) \wedge \neg(s \wedge b) \wedge \neg(a \wedge b)), \tag{6.7}$$

allocating 4 and 6 transistors for a binary and a ternary Nand, respectively [60]. Still, 20 transistors are well beyond the 10 or even 6 transistors — counting the 2 transistors

[3]In the sense of Remark 6.1.

```
    input: $s$ (mask-1), $a, b$ (simple)
    local: $s'$ (simple, initialized with 0)
    output: $o$ (simple, initialized with 0)

1: each round:
2:     $s' \leftarrow s$
3:     $o \leftarrow (\neg s \wedge a) \vee (s' \wedge b)$
4: end
```



**Algorithm 6.1:** Two-round implementation of a CMUX. The register initialization is arbitrary and the combinational logic DAG is implied by the assignments, also compare Figure 6.1(c).

for inverting $s$ for fair comparison — for a standard and a pass-gate implementation, respectively, see Section 6.3.

One way to reduce the number of gates is to use a masking register and a second round of computation to implement $f_{\text{CMUX}}$ from Equation (6.5). This is indicated by CMux-Delay in Figure 6.1(c). First, we show how to implement $f_{\text{CMUX}}$ using two clock cycles and then derive from it an efficient unclocked physical implementation with fewer gates. The former is captured by our model and the latter is not, because the model is time-discrete.

The two-round circuit is specified by Algorithm 6.1. It defines a clocked circuit by assignments of logic expressions to registers; in fact, it uses the same combinational logic DAG as Mux-Gate in Figure 6.1(a). The trick is to sequentially read $s$ from a mask-1 register, ensuring that at most one copy of $s$ can be metastable, compare the state machine in Figure 4.2(f). This guarantees that in the case of $s = \text{M}$ and $a = b = 1$, one of the And-clauses is stable 1.

**Lemma 6.4.** *Two rounds of the circuit implied by Algorithm 6.1 implement $f_{CMUX}$ from Equation* (6.5).

*Proof.* If $s \neq \text{M}$, we have $s = s'$ after round 1 and it is easy to verify that Algorithm 6.1 behaves as specified by Equation (6.5). Hence, consider $s = \text{M}$ in the remaining part of the proof. If $a \neq b$ or $a = b = \text{M}$, the output is not restricted by Equation (6.5). So consider $s = \text{M}$ and $a = b \neq \text{M}$.

If $s = \text{M}$ and $a = b = 0$, we have $o = (\neg s \wedge 0) \vee (s' \wedge 0) = 0$, as claimed. The interesting case is $s = \text{M}$ and $a = b = 1$. Here we exploit that the read phases of rounds 1 and 2 only have three possible outcomes by the state machine of mask-1 registers in Figure 4.2(f): The possible reads of $s$ are 11, 1M, and M0. Hence, in round 2, the first bit (or a stabilization of it) is stored in $s'$ and the second is read from $s$.

**Case 1** If 11 is read, $s = s' = 1$, yielding

$$o = (\neg s \wedge a) \vee (s' \wedge b) = (\neg 1 \wedge 1) \vee (1 \wedge 1) = 1. \tag{6.8}$$

**Case 2** If 1M is read, we have $s' = 1$ and obtain

$$o = (\neg s \wedge a) \vee (s' \wedge b) = (\neg s \wedge 1) \vee (1 \wedge 1) = 1. \tag{6.9}$$

**Case 3** If M0 is read, we have $s = 0$ which yields

$$o = (\neg s \wedge a) \vee (s' \wedge b) = (\neg 0 \wedge 1) \vee (s' \wedge 1) = 1. \tag{6.10}$$

As specified, the output is 1 in each case, concluding the proof. □

Observe that the circuit implied by Algorithm 6.1 uses the combinational logic from Mux-Gate which is not metastability-containing, see Figure 6.1(a). Instead of fortifying Mux-Gate with additional gates — as done in CMux-Gate in Figure 6.1(b) — Algorithm 6.1 uses a masking register and an additional clock cycle to fulfill the specification of a CMUX. This construction hence does not increase the transistor count the way CMux-Gate does and scales well with increasing bit widths $n$ of $a$ and $b$, since only the select bit needs to be stored in a masking register.

However, a hardware implementation of Algorithm 6.1 as a clocked state machine may be too slow for practical applications. Fortunately, Algorithm 6.1 has an optimized unclocked realization: The serialization of assignments in Algorithm 6.1 ensured by the two clock cycles can also be enforced by local delay constraints instead of clock cycles, see CMux-Delay in Figure 6.1(c). With a propagation delay from $s$ to the And gate with non-negated input $s$ being larger than the gate delay from $s$ to the And gate with negated input $\neg s$ — and a delay difference large enough to not observe internal metastability of the register at both And gates simultaneously — the circuit exhibits the specified behavior. Observe that this realization cannot be directly be expressed in our time-discrete circuit model from Chapter 4. However, delay lines can be avoided entirely using the completely digital transistor-level CMUXes presented below.

## 6.3 Transistor-Level Implementations

In this section, we develop efficient transistor-level implementations[4] of CMUXes. As motivated above, MUXes are ubiquitous in digital circuits and CMUXes are used in metastability-containing sorting networks [26, 100]. We hope that our implementations are a contribution to the practical feasibility of metastability-containing circuits.

We propose two CMUX implementations: CMux-A is a conservative implementation that uses 10 transistors and CMux-B uses 8 transistors at the expense of a somewhat increased peak current and a slightly deteriorated output under $s = \mathrm{M}$. Not only do we greatly improve upon the 28 transistors that Lenzen and Medina allocate for Mux-Gate [100], see above, we also have only a marginal overhead w.r.t. classical transistor-level MUX implementations of 8 transistors. We are not, however, on par with the 4 transistors of a pass-gate implementation for reasons discussed below.

Refer to Section 6.4 for a demonstration of the impact of our CMUXes. Simply using CMux-B as a drop-in replacement in metastability-containing sorting networks of Lenzen and Medina [100] and Bund et al. [26], which both heavily rely on CMUXes, reduces the respective circuit sizes by up to two thirds.

[4]We present CMOS implementations.

### 6.3.1 Transistors and CMOS Circuits

As the model from Chapter 4 does not capture transistor-level logic, the remaining part of this chapter is not covered by it. In particular, we cannot give formal correctness proofs of the proposed circuits within our model. However, our idea of adding a third, unspecified, signal to the digital world carries over to the transistor level and Complementary Metal-Oxide-Semiconductor (CMOS) logic. We need to examine the behavior of transistors under unspecified input voltages.

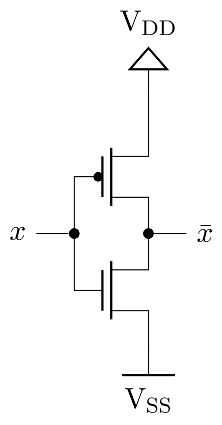


**Figure 6.3:** A CMOS inverter. The transistor in the top is a p-FET, the one in the bottom an n-FET. For $x = 0$, the p-FET closes and the n-FET opens, resulting in output 1. The opposite is the case if $x = 1$.

Consider the CMOS inverter in Figure 6.3 as example. A Field-Effect Transistor (FET) has three terminals: source, drain, and gate. There are p-FETs and n-FETs. The transistor in the top of Figure 6.3 is a p-FET, its source is connected to $V_{DD}$, logical 1, and its drain to the output on the right. The transistor in the bottom is an n-FET, its source is connected to $V_{SS}$, logical 0, and its drain to the output. Both transistors' gates are connected to the input on the left. When connecting p-FET and n-FET sources to $V_{DD}$ and $V_{SS}$, respectively, one can think of the source as the "input," the drain as the "output," and the gate as a switch controlling the source–drain resistance. This is an oversimplification, but it suffices for our purposes. We refer to standard literature [4, 131] for details.

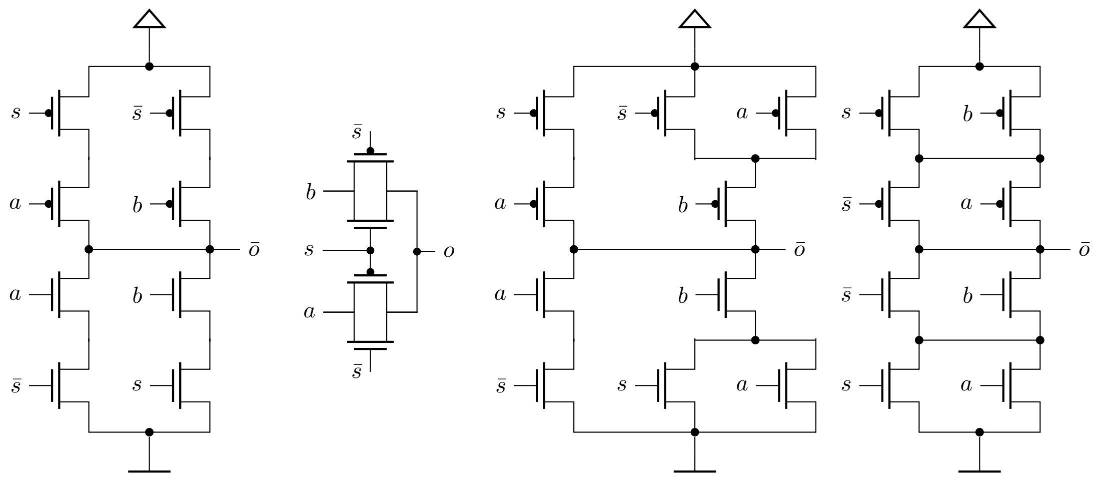
The n-FET in the bottom of Figure 6.3 has a negligibly small source–drain resistance when its gate is connected to logical 1 and an extremely high source–drain resistance when connected to logical 0. The p-FET in the top behaves the other way around: If the gate is connected to logical 0 (logical 1), it has a low (high) resistance.

For $x = 0$, the p-FET is "closed" (conductive) and the n-FET is "open" (non-conductive), i.e., there is a low-resistance path from the output to $V_{DD}$ and only one of extremely high resistance to $V_{SS}$, resulting in output $\bar{x}$.

This illustrates a fundamental CMOS concept: The logic is designed such that for any input, exactly one of $V_{DD}$ and $V_{SS}$ is connected to the output.

Next, we apply this line of reasoning to arbitrary voltages between logical 0 and 1, i.e., M. A FET exposed to a gate input of M might not be completely switched on or off, i.e., act as an unspecified source–drain resistance; this results in an unspecified output, i.e., M. Furthermore, a FET receiving M as source input has an unspecified output as well. In particular, when applying stable voltages — logical 0 or 1, i.e., $V_{SS}$ or $V_{DD}$, respectively — to a transistor's source and drain but M to its gate, it acts as an unspecified source–drain resistance. For the inverter in Figure 6.3, $x = \mathrm{M}$ means that both transistors turn into unspecified source–drain resistances, hence, the output is unspecified.

Note that our only assumption is that the behavior is unspecified in the presence



**(a)** Mux-T. **(b)** Mux-Pass. **(c)** CMux-A. **(d)** CMux-B.

**Figure 6.4:** Transistor-level MUX and CMUX implementations. Mux-T (a) and Mux-Pass (b) show conventional combinational and pass-gate MUXes, respectively. Both are vulnerable to metastability, especially Mux-Pass: $s = \text{M}$ causes an unspecified pass resistance. CMux-A (c) and CMux-B (d) implement CMUXes, i.e., metastability-containing alternatives. CMux-A requires 10 transistors and has a low peak current. CMux-B saves 2 transistors, at the expense of a slightly increased peak current in the case of $s = \text{M}$ and $a = b$.

of M, which is very conservative.

### 6.3.2 Standard Transistor-Level Multiplexers

Mux-T in Figure 6.4(a) is a standard transistor-level MUX. Under input $s = \text{M}$, all connections from $V_{\text{SS}}$ or $V_{\text{DD}}$ to the output contain a transistor with input $s$ or $\bar{s}$ at the gate, i.e., an unspecified resistance. Hence, the output may be deteriorated. This is reflected by the case of fixing $s = \text{M}$ while simultaneously switching $a = b$ from 0 to 1 or vice versa [60].

A common MUX implementation is Mux-Pass, see Figure 6.4(b). Using only four transistors, it forwards $a$ or $b$ via pass gates, i.e., without direct connections to $V_{\text{SS}}$ or $V_{\text{DD}}$. Unfortunately, pass gates have the drawback of a non-negligible pass resistance. This means that they degrade the output signal, even if all inputs are stable. Hence, chains of pass-gate MUXes should be used with caution, possibly using signal regeneration.[5] The pass-resistance problem severely intensifies in the context of metastability-containment: $s = \text{M}$ effectively turns all transistors of Mux-Pass into unspecified resistances. Simulations reveal that when fixing $s = \text{M}$ and simultaneously switching $a = b$ from 0 to 1 or vice versa — as above — Mux-Pass performs even worse than Mux-T [60]. We conclude that CMUX implementations based on pass gates need to regenerate the signal.

[5] A slightly degenerate signal can be restored by, e.g., using an inverter. This connects to $V_{\text{SS}}$ or $V_{\text{DD}}$ and serves as an amplifier, pulling the output towards logical 0 or 1.

This requires additional transistors and surrenders the original advantage, i.e., using few transistors, of a pass-gate implementation. Hence, we do not use pass gates.

### 6.3.3 Our Circuits

We propose two transistor-level implementations of a CMUX, referred to as CMux-A and CMux-B and depicted in Figures 6.4(c) and 6.4(d), respectively. Both implement a CMUX up to an inverted output[6] bit. CMux-A is a conservative implementation that requires 10 transistors. CMux-B needs only 8 transistors — the same as the conventional Mux-T — but has a slightly increased peak current and produces a slightly degraded output signal under $s = \mathrm{M}$ and $a = b$. Note that this is a very low overhead for metastability-containment.

**10 Transistors** We propose CMux-A, see Figure 6.4(c), as transistor-level implementation of a CMUX. It is not hard to verify that CMux-A works correctly if all inputs $s, a, b$ are logical 0 or 1, up to inversion of the output. Regarding correct operation under $s = \mathrm{M}$ and $a = b$, recall that a transistor with an unspecified input voltage produces an unspecified output voltage. For the proposed implementation, this implies that under $s = \mathrm{M}$, all transistors with gate input $s$ or $\bar{s}$ are to be treated as unspecified resistances.

For $s = \mathrm{M}$ and $a = b = 0$, there is a low-resistance path from $\bar{o}$ to $\mathrm{V_{DD}}$ at the top right of CMux-A, but only high-resistance paths from $\bar{o}$ to $\mathrm{V_{SS}}$, so $\bar{o} = 1 = \bar{a}$. All transistors with voltage M at the gate are bypassed and do not influence the output voltage. CMux-A behaves symmetrically if $s = \mathrm{M}$ and $a = b = 1$: There is a low-resistance path from $\bar{o}$ to $\mathrm{V_{SS}}$ but none from $\bar{o}$ to $\mathrm{V_{DD}}$. We conclude that CMux-A implements a CMUX. Simulations verify the correct behavior and establish quick and clean switching behavior under the abovementioned test case, i.e., fixing $s = \mathrm{M}$ and simultaneously switching $a = b$ from logical 0 to logical 1 and vice versa [60].

**8 Transistors** We propose CMux-B, see Figure 6.4(d), as an alternative to CMux-A. As above, it is not hard to check that CMux-B works correctly under stable inputs: If all inputs $s, a, b$ are logical 0 or 1, CMux-B outputs $\bar{a}$ if $s = 0$ and $\bar{b}$ if $s = 1$.

So consider the case $s = \mathrm{M}$ and $a = b$. Then transistors with gate input $s$ or $\bar{s}$ act as resistors of unspecified resistance, allowing a current from $\mathrm{V_{DD}}$ to $\mathrm{V_{SS}}$ along the left path of CMux-B. This results in an increased peak current under $s = \mathrm{M}$ [60].

If $a = b = 1$ the n-FETs pull the output low while the p-FETs on the left represent an unspecified resistance, which may become low enough to cause an unspecified output level. This constitutes the key difference to CMux-A which logically masks this case. To ensure correct output levels, we use transistors of double width on the right branch with inputs $a$ and $b$ [60]. Simulations confirm that CMux-B behaves correctly under input $s = \mathrm{M}$ with this tweak [60]. In the case of $a = b = 0$, CMux-B behaves symmetrically.

Altogether, CMux-B implements a CMUX, but has a higher peak current than CMux-A in the presence of a metastable select bit [60].

[6]This is not uncommon in digital circuit design, e.g., Mux-T does the same.

## 6.4 Impact on Metastability-Containing Circuits

In this section, we demonstrate how our CMUX implementations drastically reduce the complexity of existing metastability-containing circuits. This is achieved by simply using CMux-A or CMux-B as a drop-in replacement wherever a CMUX is required. We demonstrate this using the metastability-containing sorting networks for BRGC proposed by Lenzen and Medina [100] and the more recent improvement by Bund et al. [26]; using CMux-B reduces the size of 16-bit sorting networks by 69 % and 65 %, respectively.

Both papers describe 2-sort implementations that work for inputs that are either BRGC numbers or metastable superpositions of successive BRGC numbers. As BRGC only changes one bit in an increment, the latter can only become metastable at that particular bit, if implemented correctly [62]. We shed some light on this in Chapter 10.

For a 2-sort of $B$-bit BRGC numbers, Lenzen and Medina require $\mathrm{O}(B^2)$ transistors and Bund et al. $\mathrm{O}(B \log B)$. Each paper presents the transistor counts required by optimal sorting networks of 4, 7, and 10 channels for $B \in \{2, 4, 8, 16\}$ assembled from their respective 2-sort implementation. Both papers use the abovementioned 28-transistor estimate for CMux-Gate by Lenzen and Medina [100].

The proposed 2-sort implementations require a 4-CMUX, i.e., a CMUX that uses two select bits to choose one of four inputs. A 4-CMUX is a straightforward generalization of $f_{\mathrm{CMUX}}$ from Equation (6.5) and can be assembled from three CMUXes as in Figure 6.5 [100]. Note that using CMux-A or CMux-B in this two-level approach doubly negates the output and hence directly produces the desired result. Together with four transistors to invert both select bits, we require $3 \cdot 8 + 4 = 28$ transistors to implement a 4-CMUX from CMux-B,[7] much less than the $3 \cdot 28 = 84$ transistors allocated in both implementations [26, 100]. Note that in the 2-sort derived from Lenzen and Medina [100], we only need a total of two inverters for each recursion level, as both 4-CMUXes use the same select bits.

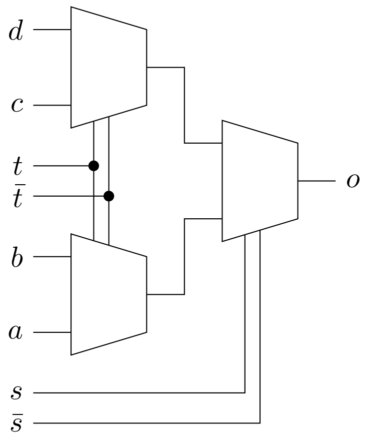


**Figure 6.5:** 4-CMUX assembled from three 2-CMUXes [100].

Table 6.1 shows how replacing the naive CMUX implementation by CMux-B from Figure 6.4(d) reduces the transistor count of metastability-containing sorting networks and gives an overview on how large these networks are.

Regarding the comparison with Lenzen and Medina, note that the authors present a polynomially as well as an exponentially sized 2-sort [100]. The exponentially sized 2-sort is more efficient for $B \in \{2, 4\}$. We always plug CMux-B into the polynomially sized circuit and compare it to the unmodified polynomially sized circuit. A comparison with the exponentially sized circuit would be obscured, because Lenzen and Medina report inconsistent transistor counts for them: The size of a 4-, 7-, and 10-sort is not exactly 5,

[7]Using CMux-A requires $3 \cdot 10 + 4 = 34$ transistors.

| bits | 4-sort | | | 7-sort | | | 10-sort | | |
|---|---|---|---|---|---|---|---|---|---|
| | [100] | we | saved | [100] | we | saved | [100] | we | saved |
| 2 | 540 | 340 | 37.0 % | 1728 | 1088 | 37.0 % | 3132 | 1972 | 37.0 % |
| 4 | 4020 | 1620 | 59.7 % | 12864 | 5184 | 59.7 % | 23316 | 9396 | 59.7 % |
| 8 | 21060 | 7060 | 66.5 % | 67392 | 22592 | 66.5 % | 122148 | 40948 | 66.5 % |
| 16 | 95460 | 29460 | 69.1 % | 305472 | 94272 | 69.1 % | 553668 | 170868 | 69.1 % |
| bits | 4-sort | | | 7-sort | | | 10-sort | | |
| | [26] | we | saved | [26] | we | saved | [26] | we | saved |
| 2 | 960 | 400 | 58.3 % | 3072 | 1280 | 58.3 % | 5568 | 2320 | 58.3 % |
| 4 | 4440 | 1640 | 63.1 % | 14208 | 5248 | 63.1 % | 25752 | 9512 | 63.1 % |
| 8 | 13920 | 4960 | 64.4 % | 44544 | 15872 | 64.4 % | 80736 | 28768 | 64.4 % |
| 16 | 37080 | 13000 | 65.0 % | 118656 | 41600 | 65.0 % | 215064 | 75400 | 65.0 % |

**Table 6.1:** Transistors saved by using CMux-B in the sorting networks of Lenzen and Medina [100] and Bund et al. [26].

10, and 29 times[8] that of a 2-sort. This, however, only makes a difference for $B \in \{2, 4\}$, as the polynomially sized 2-sort is smaller for $B \in \{8, 16\}$ and higher.

Our approach saves between 37.0 % and 69.1 % of the transistors in the described settings. The savings w.r.t. Bund et al. [26] are equally drastic: Using CMux-B reduces the size of the sorting network by 58.3 %–65.0 %, removing almost two thirds of the original circuit.

[8]The number of 2-sorts in the respective sorting networks.

# CHAPTER 7

# Combinational Circuits

In this chapter, we analyze combinational circuits. Note that in circuit design, the term "combinational circuit" refers to circuits using only combinational logic and no registers; we define combinational circuits as circuits without local registers. At first glance this may seem different, but the concept is the same: A combinational circuit does not remember previous clock cycles. As the only feedback loop in our model leads through the local registers, compare Figure 4.1(a), we can establish that property by removing them. The input and output registers required by Definition 4.3 merely are a consequence of moving the non-determinism of possible stabilization out of the combinational logic; we discuss this in Section 4.3. Hence the "registers" of a combinational circuit can simply be interpreted as input and output pins attached to the combinational logic.

We fully classify the functions implementable by combinational circuits and formally show that the output of a combinational circuit is — up to possible stabilization of metastable output bits — equivalent to its combinational logic. Furthermore, we show that it does not matter which registers are used in combinational circuits, formalizing the above claim that they may be interpreted as input and output pins.

This implies that in combinational circuits, our model essentially boils down to Sections 4.2 and 4.3: gates and combinational logic. Put differently, Kleene logic models metastability in combinational circuits. We consider this a strong indicator that our model is quite natural. In Chapter 8, we show that simple circuits are computationally equivalent to combinational circuits as well. The reason our model requires the additional complexity of registers, clock cycles, and non-determinism is to capture masking registers; we demonstrate in Chapter 8 that circuits with masking are computationally strictly more powerful than simple circuits.

**Theorem 7.1** (Combinational Circuits)**.** *Let $C$ be a combinational circuit with combinational logic DAG $G$, $m$ input registers, and $n$ output registers. Then we have*

$$C_1(\iota) = C_r(\iota) = \mathrm{Res}_{\mathrm{M}}\left(f^G(\iota)\right) \tag{7.1}$$

*for all $r \in \mathbb{N}$ and all inputs $\iota \in \mathbb{B}_{\mathrm{M}}^m$ and hence*

$$\mathrm{Fun}_C^1 = \mathrm{Fun}_C^r \tag{7.2}$$

*for all $r \in \mathbb{N}$. Furthermore, it holds for all $\iota \in \mathbb{B}_{\mathrm{M}}^m$ and all $i \in [n]$ that*

$$\begin{aligned} C_1(\iota)_i &= \{0\} &\Leftrightarrow\quad & f^G(\iota)_i = 0, & (7.3)\\ C_1(\iota)_i &= \{1\} &\Leftrightarrow\quad & f^G(\iota)_i = 1, \textit{ and} & (7.4)\\ C_1(\iota)_i &= \mathbb{B}_{\mathrm{M}} &\Leftrightarrow\quad & f^G(\iota)_i = \mathrm{M}. & (7.5)\end{aligned}$$

*Proof.* Let $\iota \in \mathbb{B}_{\mathrm{M}}^{m}$ and $r \in \mathbb{N}$ be an arbitrary input and a number of rounds. In order to show that $C_r(\iota) = \mathrm{Res}_{\mathrm{M}}(f^G(\iota))$, observe that there is an execution where we read $\iota$ from the input registers in round $r$ by the state machines in Figures 4.2(d)–4.2(f): Each input register $R$ maintains state $\iota_R$ for $r-1$ rounds and makes the uniquely determined state transition resulting in output $\iota_R$ in round $r$. The combinational logic DAG then evaluates to $f^G(\iota)$ in the $r$-th evaluation phase and the possible writes are $\mathrm{Res}_{\mathrm{M}}(f^G(\iota))$, showing that

$$C_r(\iota) \supseteq \mathrm{Res}_{\mathrm{M}}(f^G(\iota)). \tag{7.6}$$

Further observe that — also by the state machines in Figures 4.2(d)–4.2(f) — the $r$-th read phase is guaranteed to yield some $\iota' \in \mathrm{Res}_{\mathrm{M}}(\iota)$. Hence, the combinational logic DAG always evaluates to some $f^G(\iota') \in \mathrm{Res}_{\mathrm{M}}(f^G(\iota))$ by Lemma 4.2 and all possible writes are in $\mathrm{Res}_{\mathrm{M}}(f^G(\iota))$. This guarantees

$$C_r(\iota) \subseteq \mathrm{Res}_{\mathrm{M}}(f^G(\iota)). \tag{7.7}$$

As the above arguments hold for arbitrary $\iota$ and $r$, and in particular for $r = 1$, Equation (7.1), and with it Equation (7.2), follows.

Claims (7.3)–(7.5) are a direct consequence of the definition of the write phase; if the combinational logic evaluates to $x$, the possible writes are $\mathrm{Res}_{\mathrm{M}}(x)$. $\square$

Note that Equation (7.1) is independent from the register types used in the combinational circuit. This reflects the above informal statement that the registers of a combinational circuit can be interpreted as input and output pins.

**Observation 7.2.** *Let $C$ be a combinational circuit. Then $C_1$ is invariant under changing the register types used in $C$.*

In the light of Theorem 7.1, we simplify notation to

$$\mathrm{Fun}_C := \mathrm{Fun}_C^1 . \tag{7.8}$$

We establish in Chapter 8 that the computational power of combinational circuits is the same as that of simple circuits, as simple circuits can be unrolled. However, circuits that may use masking registers become strictly more powerful with each round of computation.

# CHAPTER 8

# Computational Hierarchy

In this chapter, we determine the impact of the number of rounds, of the available register types, and whether local registers are permitted on the computational power of circuits in the model from Chapter 4. Recall from Definition 4.6 that $\mathrm{Fun}_M^r$, $\mathrm{Fun}_S^r$, and $\mathrm{Fun}_C^r$ denote the functions implementable using $r \in \mathbb{N}$ rounds in arbitrary circuits, in simple circuits, and in combinational circuits, respectively, where simple circuits are restricted to simple registers and combinational circuits have no local registers. Further recall that the number of rounds is irrelevant in combinational circuits, i.e., that $\mathrm{Fun}_C = \mathrm{Fun}_C^1 = \mathrm{Fun}_C^r$ by Theorem 7.1. The main results are the following.

(1) Even in the presence of metastability, circuits restricted to simple registers can be unrolled. This means that $r \in \mathbb{N}$ copies of the combinational logic can be arranged to implement $r$ rounds of the original circuit in a single round. In terms of computable functions, this means that $\mathrm{Fun}_S := \mathrm{Fun}_S^1 = \mathrm{Fun}_S^r$. We discuss this in Section 8.1.

(2) Combinational circuits are exactly as powerful as simple circuits: $\mathrm{Fun}_C = \mathrm{Fun}_S$. This is discussed in Section 8.2.

(3) With masking registers, however, unrolled circuits are not equivalent to multiple rounds of computation anymore. Regarding computable functions, we show in Section 8.3 that this leads to a strict inclusion: $\mathrm{Fun}_M^r \subsetneq \mathrm{Fun}_M^{r+1}$ for all $r \in \mathbb{N}$.

Together with Corollary 4.8, we obtain the following hierarchy:

$$\mathrm{Fun}_C = \mathrm{Fun}_S = \mathrm{Fun}_M^1 \subsetneq \mathrm{Fun}_M^2 \subsetneq \mathrm{Fun}_M^3 \subsetneq \cdots . \tag{8.1}$$

We believe this to make a strong case for further pursuing masking registers in research regarding metastability-containing circuits.

## 8.1 Simple Circuits

It is folklore that binary-valued synchronous circuits can be unrolled such that the output after $r \in \mathbb{N}$ clock cycles of the original circuit is equal to the output after a single clock cycle of the unrolled circuit. Theorem 8.1 states that this result also holds in presence of potentially metastable simple registers. Carefully note that this result — defying intuition — does not carry over to circuits with masking registers, see Theorem 8.3.

**Theorem 8.1** (Unrolling). *Given a simple circuit $C$ and $r \in \mathbb{N}$, one can construct a simple circuit $C'$ such that $C'_1 = C_r$. In particular, if $r$ rounds of $C$ implement $f$, a single round of $C'$ implements $f$. Hence, it holds that*

$$\forall r \in \mathbb{N}\colon \quad \mathrm{Fun}_S^1 = \mathrm{Fun}_S^r . \tag{8.2}$$

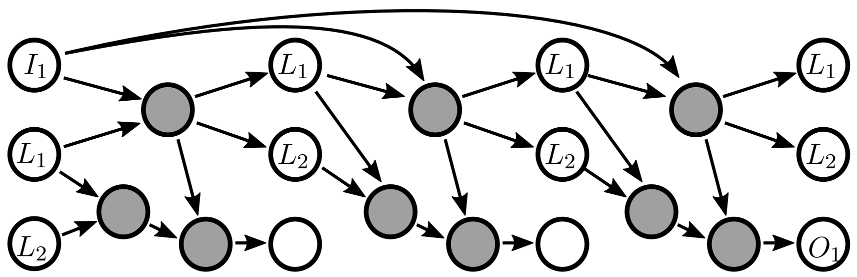


**Figure 8.1:** Unrolling three rounds of the circuit in Figure 4.4 with three gates (gray), and four registers (white). Local registers ($L_1$ and $L_2$) become Id gates and early output is ignored.

*Proof.* The claim is trivial for $r = 1$, hence suppose $r \geq 2$. We construct a circuit $C'$ with $C'_1(\iota) = C_2(\iota)$. Given $C$, we construct $C'$ as follows, compare Figure 8.1. Let $G$ be the combinational logic DAG of $C$, consider two copies $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$ of $G$, and let $G' = (V_1 \dot\cup V_2, A_1 \dot\cup A_2)$ be the combinational logic DAG of $C'$, up to the following modifications.

(1) Every input register $I$ of $C$ corresponds to input nodes $v_1^I \in V_1$ and $v_2^I \in V_2$. Identify $\{v_1^I, v_2^I\}$ to a single input node in $G'$ (compare $I_1$ in Figure 8.1) and associate it with a new input register in $C'$; apply this to all input registers of $C$.

(2) In order to ignore "early" output, replace each output node in $G_1$ corresponding to an output register in $C$ with a gate that has one input and whose output is ignored (like the first two copies of $O_1$ in Figure 8.1).

(3) The remaining input and output nodes are associated with local registers. Each local register $L$ of $C$ corresponds to exactly one output node $v_1^L \in V_1$ and one input node $v_2^L \in V_2$. Identify $\{v_1^L, v_2^L\}$ to an Id gate in $G'$ that simply forwards its input (the center copies of $L_1$ and $L_2$ in Figure 8.1).

Associate the $k$ remaining input nodes of $G_1$ and output nodes of $G_2$ with local registers, maintaining the original mapping. Observe that $G'$ has $n$ input, $m$ output, and $k$ local registers; we rule that all registers are simple. Define the initial state of $C'$ as that of $C$.

To check that one round of $C'$ is equivalent to two rounds of $C$, let $\iota \in \mathbb{B}_{\mathrm{M}}^m$ be an input, $s_0$ the initial state of both $C$ and $C'$ w.r.t. input $\iota$, and $x_0 = \mathrm{In}(s_0) \circ \mathrm{Loc}(s_0)$ the initial state of the non-output registers. As all registers are simple, we have $\mathrm{Read}^{C'}(x_0) = \{x_0\}$ by Observation 4.4 and hence, by construction of $G'$,

$$\mathrm{Eval}^{C'}(s_0) = \left\{f^{G'}(x_0)\right\} = \left\{f^G\left(\iota \circ \mathrm{Loc}\left(f^G(x_0)\right)\right)\right\}. \tag{8.3}$$

In $C$, we have $\mathrm{Write}^C(s_0) = \mathrm{Res}_{\mathrm{M}}(f^G(x_0))$ by Lemma 4.7. Recall that by the state machine in Figure 4.2(d), simple registers do not change their state when read. Thus, when $C$ is in some state $s_1 \in S_1^C$ after the first round, we have that

$$\begin{aligned} \mathrm{Read}^C(s_1) &= \{\iota \circ \mathrm{Loc}(s_1)\} && (8.4)\\ &\subseteq \left\{\iota \circ \mathrm{Loc}(x) \mid x \in \mathrm{Write}^C(s_0)\right\} && (8.5)\\ &\subseteq \mathrm{Res}_{\mathrm{M}}\left(\iota \circ \mathrm{Loc}\left(f^G(x_0)\right)\right), && (8.6)\end{aligned}$$

using Observation 4.4. By Lemma 4.2, we obtain

$$\mathrm{Eval}^C(s_1) \subseteq \mathrm{Res}_{\mathrm{M}}\left(f^G\left(\iota \circ \mathrm{Loc}\left(f^G(x_0)\right)\right)\right). \tag{8.7}$$

This means that the second evaluation phase of $C$ yields a stabilization of the first evaluation phase of $C'$, because the write phase allows for arbitrary stabilization. On the other hand, it is possible that no stabilization happens in $C$, as the unstabilized $\iota \circ \mathrm{Loc}(f^G(x_0)) \in \mathrm{Eval}^C(s_1)$. Together, we have $S_2^C = S_1^{C'}$ and $C_1'(\iota) = C_2(\iota)$ follows.

An inductive application of the above arguments yields the claims. □

Naturally, the unrolled circuit is larger than the original and a physical implementation needs longer clock cycles, as the increase in delay is linear in $r$. However, the point is that adding rounds does not affect the computational power of simple circuits. In the light of Theorem 8.1, we use

$$\mathrm{Fun}_S := \mathrm{Fun}_S^1 \tag{8.8}$$

in the following.

## 8.2 Combinational Circuits

As simple circuits can be unrolled, local registers do not contribute to their computational power. Hence, simple circuits are exactly as powerful as combinational circuits.

**Theorem 8.2.** *We have*

$$\mathrm{Fun}_S = \mathrm{Fun}_C\,. \tag{8.9}$$

*Proof.* Consider $f \in \mathrm{Fun}_C$. By Observation 7.2, there is a simple, combinational circuit $C$ that implements $f$. As $C$ is simple, $f \in \mathrm{Fun}_S$ and hence $\mathrm{Fun}_C \subseteq \mathrm{Fun}_S$.

On the other hand, consider $f \in \mathrm{Fun}_S$. By Theorem 8.1, there is a simple circuit $C$ that implements $f$ in a single round. As $C$ only uses one round, its local registers are only read once. Hence, replacing each local register with a Const0 or a Const1 gate, depending on its initialization, yields a combinational circuit that implements $f$. We conclude $\mathrm{Fun}_S \subseteq \mathrm{Fun}_C$ and the claim follows. □

## 8.3 Arbitrary Circuits

As discussed above, additional rounds make no difference in terms of computability for simple and combinational circuits; the corresponding hierarchies collapse into $\mathrm{Fun}_S$. In the presence of masking registers, however, unrolled circuits are not equivalent to multiple rounds of computation: As detailed in Section 8.1, unrolling requires an input register to be connected to each copy of the combinational logic. Since a masking register in

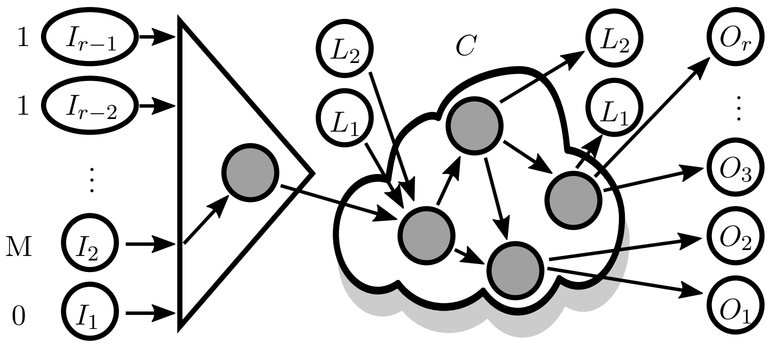


**Figure 8.2:** Simulating a masking register with a selector.

state M may be read as M, all these copies may receive M as input. This, however, does not happen when reading from the masking register sequentially: A masking register guarantees that M is read at most once, compare Figure 4.2.

Below, we demonstrate that this leads to a strict inclusion: $\mathrm{Fun}_M^r \subsetneq \mathrm{Fun}_M^{r+1}$ for all $r \in \mathbb{N}$. We show this using a metastability-containing fan-out buffer specified by Equation (8.10). It creates $r$ copies of its input bit, at most one of which is permitted to become metastable:

$$f(x) := \begin{cases} \{x^r\} & \text{if } x \neq \mathrm{M} \text{ and} \\ \bigcup_{i\in[r]} \mathrm{Res}_{\mathrm{M}}\left(0^i \mathrm{M} 1^{r-i-1}\right) & \text{otherwise.} \end{cases} \tag{8.10}$$

**Theorem 8.3.** *It holds that*

$$\forall r \in \mathbb{N}: \quad \mathrm{Fun}_M^r \subsetneq \mathrm{Fun}_M^{r+1}. \tag{8.11}$$

*Proof.* Fix $2 \leq r \in \mathbb{N}$ and consider $f$ from Equation (8.10). We first show $f \in \mathrm{Fun}_M^r$, and then that $f \notin \mathrm{Fun}_M^{r-1}$.

First observe that $f$ is implemented by $r$ rounds of the circuit $C$ which uses a mask-0 input register $R_{r-1}$ and a chain of local registers $R_{r-2}, \ldots, R_0$. In each round, the value read from register $R_{i+1}$, $i \in [r-1]$, is copied to $R_i$ and output register $O_i$, $i \in [r]$, gets the value read from $R_i$. Observe that the specification of a mask-0 register is such that, given an initial state, $r$ reads (and possibly stabilization in the write phase) may return exactly the sequences specified in Equation (8.10). Since $C$ faithfully copies these values, it follows that $f = C_r \in \mathrm{Fun}_M^r$.

Below, we show that $f \notin \mathrm{Fun}_M^{r-1}$, for which we use the following subcircuits:

**$r$-round counters** take no input and have $r$ outputs, such that the $(i-1)$-th output is 1 in round $1 \leq i \leq r$ and 0 else. This is implemented by a linear chain of local registers $R_i$, $i \in [r]$, where each $R_i$ is copied to $R_{i+1}$ for $i \in [r-1]$, $R_0$ is initialized to 1 and all others to 0. Output register $O_i$, $i \in [r-1]$, is fed the Xor of $R_i$ and $R_{i+1}$, and $R_{r-1}$ is copied to $O_{r-1}$.

**$r$-round selectors** take $r$ inputs $x_i$, $i \in [r]$, and have one output $O$, such that the state of $O$ in round $1 \leq i \leq r$ is in $\mathrm{Res}_{\mathrm{M}}(x_{i-1})$, i.e., holds a copy of $x_{i-1}$. This is achieved using an $r$-round counter and feeding the AND of $x_i$ and the $i$-th counter output into an $r$-ary OR, the output of which is written to $O$.

We claim that $f \notin \mathrm{Fun}_M^{r-1}$. Assume for contradiction that there is a circuit $C$ such that $r-1$ rounds of $C$ implement $f$, i.e., $C_{r-1} \subseteq f$. We derive a contradiction by simulating the behavior of $C$ in a circuit $C'$ with $r-1$ simple input registers, which may initially hold any possible sequence of values read from the input register of $C$ in $r-1$ rounds.

By Corollary 4.9, we may assume w.l.o.g. that all non-input registers of $C$ are simple. If the only input register of $C$ is also simple as well, we may unroll $C$ into $\bar{C}$ by Theorem 8.1. Applying Lemma 4.7 to $\bar{C}$ yields $\mathrm{M}^r \in \bar{C}_1(\mathrm{M}) = C_{r-1}(\mathrm{M}) \notin f(\mathrm{M})$, i.e., violating the specification. Hence, suppose the input register of $C$ is a masking register.

Consider the case that the input register is a mask-0 register and compare Figure 8.2. Define $C'$ as a copy of $C$, except that $r-1$ simple input registers serve as input to an $(r-1)$-round selector. This compound represents the only input register $R$ of $C$: Every gate or output node driven by $R$ in $C$ is instead connected to the selector's output in $C'$.

A surjective mapping of executions of $C'$ with inputs restricted to all possible sequences of $r-1$ reads from $R$ in state M, i.e., restricted to inputs

$$\left\{0^{r-1}\right\} \cup \left\{0^i \mathrm{M} 1^{r-i-2} \mid i \in [r-1]\right\}, \tag{8.12}$$

to executions of $C$ is defined as follows. We interpret the selector's output in round $r$ as the value read from $R$ in round $r$ and "copy" the remaining execution of $C'$ (without inputs and the selector) to obtain a complete execution of $C$. As our restriction of the inputs reflects exactly the possible reads of a mask-0 register, see the state machine in Figure 4.2(e), the result always is a feasible execution of $C$ with input M.

By Theorem 8.1, we may w.l.o.g. assume that a single round of $C'$ implements $f$. Consider the sequence of $C'$-inputs from $0^{r-1}$ to $1^{r-1}$ in which we flip the bits one by one from right to left, from 0 to 1, with some parallels to Fischer et al. [52]. By the pigeon hole principle, there must be some $1 \le \bar{r} \le r-1$ so that two output bits of $C'$ change compared to $\bar{r}-1$. Since, when fixing the other input bits, two outputs $\ell \neq \ell'$ depend on the $\bar{r}$-th input bit from the right and $C'$ only uses simple registers, we have by Lemma 4.7 that

$$\mathrm{MM} \in \mathrm{Write}^{C'}\left(0^{r-\bar{r}-1}\mathrm{M}1^{\bar{r}-1} \circ x_0\right)_{\ell,\ell'}, \tag{8.13}$$

where $x_0$ is the initialization of the local registers. Hence, $\ell$ and $\ell'$ can become metastable in the same execution of $C'$. We map this execution to an execution of $C$, in which the corresponding output registers attain the same state (i.e., two of them are M) after $r-1$ rounds. This contradicts the assumption that $r-1$ rounds of $C$ implement $f$ from Equation (8.10).

The above argument covers the case that the input register is a mask-0 register. A mask-1 register is handled analogously by replacing Equation (8.12) with all possible reads of a mask-1 register, i.e., by

$$\left\{1^{r-1}\right\} \cup \left\{1^i \mathrm{M} 0^{r-i-2} \mid i \in [r-1]\right\}, \tag{8.14}$$

and adapting the subsequent arguments accordingly.

From the contradictions derived above, we conclude that $C_{r-1} \not\subseteq f$. This implies that $f \notin \mathrm{Fun}_M^{r-1}$ and $\mathrm{Fun}_M^{r-1} \neq \mathrm{Fun}_M^r$. As $r \geq 2$ was arbitrary and, by Observation 4.12, $\mathrm{Fun}_M^{r-1} \subseteq \mathrm{Fun}_M^r$, this concludes the proof. $\square$

# CHAPTER 9

# Simple and Combinational Circuits

In this chapter, we fully classify $\mathrm{Fun}_S = \mathrm{Fun}_C$. By Theorems 8.1 and 8.2, our results carry over to an arbitrary number of rounds in simple circuits and to combinational circuits in particular. We present sufficient and necessary conditions for a function to be implementable with a simple or combinational circuit.

Using this classification, we demonstrate how to take an arbitrary Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$ and extend it to the most restrictive specification $[f]_\mathrm{M}\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$, the *metastable closure of* $f$*,* that is implementable in simple and combinational circuits.

The way to use this when designing metastability-containing circuits is to start with a function $f$ required as component, "lift" it to $[f]_\mathrm{M}$, and check whether $[f]_\mathrm{M}$ is restrictive enough for the application at hand. If it is, one can work on an efficient implementation of $[f]_\mathrm{M}$, otherwise a new strategy, possibly involving masking registers, must be devised.

## 9.1 Natural Subfunctions

Let $C$ be a simple circuit. From Corollary 4.10 and Observation 4.11, we know that $C_1$, the set of possible circuit outputs after a single round, has three properties: (1) its output can be specified bit-wise, (2) each output bit is either 0, 1, or completely unspecified, and (3) stabilizing a partially metastable input restricts the set of possible outputs. Hence $C_1$ — and by Theorems 8.1 and 8.2 all simple and combinational circuits — can be represented in terms of bit-wise Karnaugh maps with values "$\{0\}, \{1\}, \mathbb{B}_\mathrm{M}$" instead of "$0, 1, \mathrm{D}$" (D for "don't care"); as this is equivalent to combinational logic by Theorem 7.1, we may also use "$0, 1, \mathrm{M}$." We call such functions *natural* and show below that $f \in \mathrm{Fun}_S$ if and only if $f$ has a natural subfunction.

**Definition 9.1** (Natural Function)**.** *The function* $f\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$ *is* natural *if and only if it is* bit-wise, closed, *and* specific:

**Bit-wise** *The components* $f_1, \dots, f_n$ *of* $f$ *are independent:*

$$f(x) = f_1(x) \times \dots \times f_n(x). \tag{9.1}$$

**Closed** *Each component of* $f$ *is specified as either* 0*, as* 1*, or completely unspecified:*

$$\forall x \in \mathbb{B}_\mathrm{M}^m\colon \quad f(x) \in \{\{0\}, \{1\}, \mathbb{B}_\mathrm{M}\}^n\,. \tag{9.2}$$

**Specific** *Stabilizing partially metastable input does not destabilize the output of* $f$*:*

$$\forall x \in \mathbb{B}_\mathrm{M}^m\colon \quad x' \in \mathrm{Res}(x) \Rightarrow f(x') \subseteq f(x). \tag{9.3}$$

Suppose we ask whether a function $f$ is implementable with a simple or combinational circuit, i.e., if $f \in \mathrm{Fun}_S$. If there is a simple circuit that implements $f$, we can unroll it by Theorem 8.1 and may thus assume w.l.o.g. that $C_1 \subseteq f$. Hence, Corollary 4.10 and Observation 4.11 state a necessary condition for $f \in \mathrm{Fun}_S$: $f$ must have a natural subfunction. Theorem 9.2 establishes that this condition is sufficient, too.

The main technique of Theorem 9.2 — covering all prime implicants — is mentioned by Huffman [74] and formally shown to prevent logic hazards by Eichelberger [43], who also uses Kleene logic [90]. This is owed to the fact that Eichelberger uses Kleene logic to model gate behavior under switching inputs [43]. As what we refer to as "M" includes arbitrary signal behavior over time — see Chapter 4 — and hence also switching signals, this analogy is not surprising. For the sake of self-containment and as our notation and model are quite different, we state and prove Theorem 9.2.

**Theorem 9.2.** *Let $g\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$ be a function. Then it holds that $g \in \mathrm{Fun}_S$ if and only if $g$ has a natural subfunction.*

*Proof.* Regarding the "$\Rightarrow$" direction, we have a simple circuit $C$ such that $C_1 \subseteq g$ by Theorem 8.1. $C_1$ is bit-wise and closed by Corollary 4.10 and specific by Observation 4.11. Hence, $C_1$ is a natural subfunction of $g$.

We proceed with the "$\Leftarrow$" direction. Let $f \subseteq g$ be a natural subfunction of $g$; we construct a circuit $C$ that implements $f$ in one round. As $f$ is bit-wise, we may w.l.o.g. assume that $n = 1$. If $f = \mathrm{const}_{\{0\}}$ or $f = \mathrm{const}_{\mathbb{B}_{\mathrm{M}}}$, let $C$ be the circuit whose output register is driven by a Const0 gate; if $f = \mathrm{const}_{\{1\}}$, use a Const1 gate.

Otherwise, consider $f_{\mathbb{B}}\colon \mathbb{B}^m \to \{\{0\},\{1\}\}$ given by

$$f_{\mathbb{B}}(x) = \begin{cases} \{0\} & \text{if } f(x) = \{0\} \text{ or } f(x) = \mathbb{B}_{\mathrm{M}} \text{ and} \\ \{1\} & \text{if } f(x) = \{1\}. \end{cases} \tag{9.4}$$

We call a partial variable assignment $A$ that implies $f_{\mathbb{B}}(x) = \{1\}$ for all $x$ obeying $A$ an *implicant of $f_{\mathbb{B}}$;* if the number of variables fixed by $A$ is minimal w.r.t. $A$ being an implicant, we refer to $A$ as a *prime implicant of $f_{\mathbb{B}}$* [29].

Construct the combinational logic DAG of $C$ in a Disjunctive Normal Form (DNF) fashion: Use an And gate[1] for each prime implicant of $f_{\mathbb{B}}$, with inputs connected to the respective, possibly negated, input registers. The respective outputs feed an Or gate[2] driving the circuit's only output register. See Figure 9.1 for an illustration. $C$ has $m$ input registers, no local registers, and one output register; all registers are simple as required. The initialization of the output register is arbitrary. By construction, we have that

$$\forall x \in \mathbb{B}^m\colon \quad C_1(x) = f_{\mathbb{B}}(x) \subseteq f(x). \tag{9.5}$$

To see that $C_1 \subseteq f$, consider $x \in \mathbb{B}_{\mathrm{M}}^m \setminus \mathbb{B}^m$ and make a case distinction.

**Case 1** If $f(x) = \mathbb{B}_{\mathrm{M}}$, then trivially $C_1(x) \subseteq f(x)$.

[1]If no And gate of sufficient fan-in is available, use a tree of And gates. For the sake of presentation, and because it is equivalent to the tree, we assume an And gate of sufficient fan-in is available.

[2]As above, we assume an Or gate of sufficient fan-in is available.

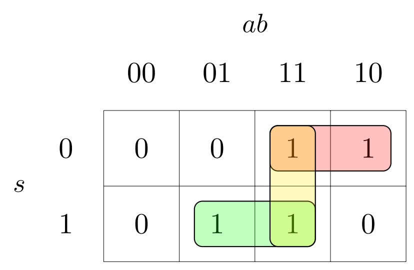


**Figure 9.1:** Karnaugh map of a MUX. Mux-Gate from Chapter 6 covers the prime implicants $sa = 01$ and $sb = 11$. The metastability-containing implementation CMux-Gate additionally covers $ab = 11$; this corresponds to the construction in the proof of Theorem 9.2.

**Case 2** If $f(x) = \{0\}$, we have for all $x' \in \operatorname{Res}(x)$ that $f(x') = \{0\} = f_{\mathbb{B}}(x')$ by Equation (9.3), as $f$ is specific. Thus, for each such $x'$, all And gates, corresponding to prime implicants, output 0. Furthermore, under input $x$ and for each And gate, there must be at least one input that is stable 0: Otherwise, there would be some $x' \in \operatorname{Res}(x)$ making one And gate, i.e., prime implicant, output 1, resulting in $C_1(x') = \{1\}$ and contradicting Equation (9.5). By our definition of gate behavior, this entails that all And gates output 0 for all $x' \in \operatorname{Res}_{\mathrm{M}}(x)$ as well, and hence

$$C_1(x) = \{0\} = f(x). \tag{9.6}$$

**Case 3** If $f(x) = \{1\}$, all $x' \in \operatorname{Res}(x)$ have $f(x') = f_{\mathbb{B}}(x') = \{1\}$ by Equation (9.3). Thus, $f_{\mathbb{B}}$ outputs $\{1\}$ independently from the metastable bits in $x$, and there is a prime implicant of $f_{\mathbb{B}}$ which relies only on stable bits in $x$. By construction, some And gate in $C$ implements that prime implicant. This And gate receives only stable inputs from $x$, and hence outputs a stable 1. The Or gate receives that 1 as input and, by definition of gate behavior, outputs stable 1. We conclude that

$$C_1(x) = \{1\} = f(x). \tag{9.7}$$

As $f$ is closed, the above case distinction is exhaustive. The claim follows as one round of $C$ implements $f$. □

As an example, consider a binary single-bit MUX specification $f\colon \mathbb{B}^3 \to \mathbb{B}$,

$$f(s,a,b) = \begin{cases} a & \text{if } s = 0 \text{ and} \\ b & \text{otherwise,} \end{cases} \tag{9.8}$$

and its Karnaugh map in Figure 9.1. A DNF implementation that is unaware of metastability only has to cover all ones in the Karnaugh map with implicants. This is achieved by implementing the prime implicants $sa = 01$ and $sb = 11$, which is exactly what Mux-Gate from Chapter 6 does. Applying the construction of the proof of Theorem 9.2, on the other hand, requires implementing all prime implicants, i.e., to implement $ab = 11$ in addition to $sa = 01$ and $sb = 11$. This yields CMux-Gate from Chapter 6, a CMUX implementation.

## 9.2 Metastable Closure

Consider a Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$. Lift the definition of $f$ to $[f]_\mathrm{M}\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$, dealing with metastable inputs analogously to gate behavior in Section 4.2: Whenever all metastable input bits together can influence the output, specify the output as "anything in $\mathbb{B}_\mathrm{M}$." We call $[f]_\mathrm{M}$ the *metastable closure of $f$*, and argue below that $[f]_\mathrm{M} \in \mathrm{Fun}_S$. For $f\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$, i.e., for more flexible specifications, $[f]_\mathrm{M}$ is defined analogously. Note that the concept of the metastable closure is equivalent to the absence of Eichelberger's logic hazards [43].

**Definition 9.3** (Metastable Closure)**.** *For a function $f\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$, we define its* metastable closure *$[f]_\mathrm{M}\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$ component-wise for $i \in [n]$ by*

$$[f]_\mathrm{M}(x)_i := \begin{cases} \{0\} & \text{if } \forall x' \in \mathrm{Res}_\mathrm{M}(x)\colon\ f(x')_i = \{0\}, \\ \{1\} & \text{if } \forall x' \in \mathrm{Res}_\mathrm{M}(x)\colon\ f(x')_i = \{1\}, \text{ and} \\ \mathbb{B}_\mathrm{M} & \text{otherwise.} \end{cases} \tag{9.9}$$

*We generalize this to Boolean functions. For $f\colon \mathbb{B}^m \to \mathbb{B}^n$, we define $[f]_\mathrm{M}\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$ as*

$$[f]_\mathrm{M}(x)_i := \begin{cases} \{0\} & \text{if } \forall x' \in \mathrm{Res}(x)\colon\ f(x')_i = 0, \\ \{1\} & \text{if } \forall x' \in \mathrm{Res}(x)\colon\ f(x')_i = 1, \text{ and} \\ \mathbb{B}_\mathrm{M} & \text{otherwise.} \end{cases} \tag{9.10}$$

**Example 9.4.** *The metastable closure of a binary MUX specification from Equation* (9.8) *is exactly the specification of a CMUX from Equation* (6.5)*.*

Observe that Equation (9.10) is equivalent to the behavior we demand from gates in Section 4.2 when identifying 0 with $\{0\}$, 1 with $\{1\}$, and M with $\mathbb{B}_\mathrm{M}$, respectively. Furthermore, note that for $f\colon \mathbb{B}^m \to \mathbb{B}^n$, the metastable closure is very closely related to the metastable superposition:

$$[f]_\mathrm{M}(x) = \mathrm{Res}_\mathrm{M}\left( \bigoplus_{x' \in \mathrm{Res}(x)} f\left(x'\right) \right). \tag{9.11}$$

Most importantly, observe that $[f]_\mathrm{M}$ is bit-wise, closed, and specific by construction; hence $[f]_\mathrm{M}$ is natural.

**Observation 9.5.** $[f]_\mathrm{M} \in \mathrm{Fun}_S$ *for all $f\colon \mathbb{B}^m \to \mathbb{B}^n$ and all $f\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$.*

An immediate consequence of Observation 9.5, Theorem 8.1, and Theorem 8.2 for the construction of circuits is that, given an arbitrary Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$, there are simple and combinational circuits that implement $[f]_\mathrm{M}$ in a single round.

**Observation 9.6.** *For $f\colon \mathbb{B}^m \to \mathbb{B}^n$, Theorem 9.2 shows that $[f]_{\mathrm{M}}$ is the most restrictive extension of $f$ implementable with simple or combinational circuits: By Equation* (9.3)*, any natural extension $g$ of $f$ must satisfy*

$$\forall x \in \mathbb{B}_{\mathrm{M}}^m, \forall i \in [n]\colon \qquad \bigcup_{x' \in \mathrm{Res}(x)} f(x')_i \subseteq g(x)_i, \tag{9.12}$$

*and thus that*

$$\exists x', x'' \in \mathrm{Res}(x)\colon\ f(x')_i \neq f(x'')_i \quad \Rightarrow \quad g(x)_i = \mathbb{B}_{\mathrm{M}} \tag{9.13}$$

*by Equation* (9.2)*.*

In order to show that a function is not implementable with simple registers only, it suffices to show that it violates the preconditions of Theorem 9.2, i.e., that it has no natural subfunction.

**Example 9.7.** *Consider $f\colon \mathbb{B}_{\mathrm{M}}^2 \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^2)$ with*

$$f(x) := \mathrm{Res}_{\mathrm{M}}(x) \setminus \{\mathrm{MM}\}. \tag{9.14}$$

*This function specifies to copy a 2-bit input; if there is metastability in the input, it may be propagated to at most one output. This specification does not contradict the impossibility results discussed in Chapter 5, but still, no circuit without masking registers implements $f$, i.e., $f \notin \mathrm{Fun}_{\mathcal{S}}$.*

*Proof.* Assume for the sake of contradiction that $f \in \mathrm{Fun}_{\mathcal{S}}$, i.e., that $f$ has a natural subfunction $g \subseteq f$ by Theorem 9.2.

As specified in Equation (9.14), $f(00)_1 = \{0\}$ and $f(11)_1 = \{1\}$. Since $g \subseteq f$ and $g(x) \neq \emptyset$ because $g$ is closed, we have $g(00)_1 = \{0\}$ and $g(11)_1 = \{1\}$. The fact that $g$ is specific implies that $g(00)_1 \cup g(11)_1 \subseteq g(\mathrm{MM})_1$, i.e., $\{0,1\} \in g(\mathrm{MM})_1$. This in turn means that $g(\mathrm{MM})_1 = \mathbb{B}_{\mathrm{M}}$, because $g$ is closed. Furthermore, we know that $g$ is bit-wise, so $g = g_1 \times g_2$ with $g_1(00) = \{0\}$, $g_1(11) = \{1\}$, and $g_1(\mathrm{MM}) = \mathbb{B}_{\mathrm{M}}$. Analogously, $g_2(00) = \{0\}$, $g_2(11) = \{1\}$, and $g_2(\mathrm{MM}) = \mathbb{B}_{\mathrm{M}}$.

Since $g$ is bit-wise, $g(\mathrm{MM}) = g_1(\mathrm{MM}) \times g_2(\mathrm{MM}) \ni \mathrm{MM}$, but $\mathrm{MM} \notin f(\mathrm{MM})$, contradicting the assumption $g \subseteq f$. As we did not make any restrictions regarding $g$, this holds for all natural subfunctions of $f$. It follows that $f \notin \mathrm{Fun}_{\mathcal{S}}$. □

## 9.3 Open Problem

Theorem 9.2 is useful for checking if a combinational or simple circuit implementing some function exists. Furthermore, its proof is constructive, providing an indication on where to start with the design of such a circuit. However, it is well-known that covering all prime implicants can be costly as there may be exponentially many [29]. While efficient metastability-containing implementations for some problems are known [26, 100], there is a fundamental open question: What is the overhead of implementing the metastable closure?

More precisely, let $f\colon \mathbb{B}^m \to \mathbb{B}^n$ be a Boolean function and consider the smallest possible combinational circuit, in terms of the number of gates, that implements

$$\bar{f}\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n) \tag{9.15}$$

$$\bar{f}(x) = \begin{cases} \{f(x)\} & \text{if } x \in \mathbb{B}^m \text{ and} \\ \mathbb{B}_{\mathrm{M}} & \text{otherwise.} \end{cases} \tag{9.16}$$

What is the worst-case overhead of implementing $[f]_{\mathrm{M}}$ instead of $\bar{f}$ in a combinational circuit w.r.t. a fixed set of gates?

As an example, the combinational circuits Mux-Gate and CMux-Gate in Chapter 6 which implement $f_{\mathrm{MUX}}$ and $f_{\mathrm{CMUX}} = [f_{\mathrm{MUX}}]_{\mathrm{M}}$, respectively. When restricting the gate fan-in to 2, CMux-Gate uses six gates[3] and Mux-Gate only uses four[4] but does not implement $[f_{\mathrm{MUX}}]_{\mathrm{M}}$. Is there a gate-level CMUX with five or four gates when restricted to Const0, Const1, Not, binary Or, and binary And?

[3]CMux-Gate uses one Not, three And, and two Or gates.

[4]Mux-Gate has one Not, two And, and one Or gate.

# CHAPTER 10

# Clock Synchronization

We establish the implementability of the fault-tolerant clock-synchronization algorithm by Lundelius Welch and Lynch [106] with a deterministic correctness guarantee, despite the unavoidable [108] presence of metastable upsets. As we do not require synchronizers, yet tolerate metastable upsets, this has two important consequences:

(1) Synchronization delay does not impose a fundamental limit on the operating frequency of clock-synchronization algorithms in hardware.

(2) An implementation of the proposed algorithm can, in principle, remove the risk of metastability from communication across clock domains altogether, as communication between synchronized clock domains does not incur metastable upsets.

We demonstrate that a variety of metastability-containing combinational circuits — referred to as *components* — can be implemented. Due to the machinery established in the preceding chapters, this is possible with simple checks, usually using Observation 9.5. We remark that since the paper this chapter is based on [55] appeared, efficient implementations of some of the components described below have been proposed [26, 62, 100]. The list of components we present is by no means an exhaustive list on what computations can be realized in a metastability-containing way, yet it permits implementing a highly non-trivial application: a hardware implementation of the fault-tolerant clock synchronization algorithm of Lundelius Welch and Lynch [106].

The algorithm of Lundelius Welch and Lynch is widely applied, e.g., in the Time-Triggered Protocol (TTP) [92, 93, 94] — in the context of the TTP, this is commonly referred to as the *fault-tolerant average algorithm,* which uses the strategy of Lundelius Welch and Lynch, see Kopetz and Ochsenreiter [94] — and FlexRay [17] clock-synchronization protocols. Kopetz et al. predict an achievable precision in the order of microseconds for software–hardware based implementations of TTP [93]. High operating frequencies, however, ultimately require a pure hardware implementation: 1 GHz clocks, for example, only leave very few CPU cycles per pulse. Kinali et al. present an FPGA implementation that uses synchronizers instead of metastability-containing techniques [83]. We are not aware of a metastability-containing implementation of the clock-synchronization algorithm as a whole.

All known implementations synchronize potentially metastable inputs from the TDCs *before* computations. This technique becomes less reliable with increasing operating frequencies since less time is available for metastability resolution. Furthermore, low temperatures and supply voltages intensify this issue [14]. Moreover, classical bounds for the MTBF regarding metastable upsets in synchronizers assume a uniform distribution of input transitions [13, 89]; this is not guaranteed to be the case in clock synchronization, since the goal is to align clock ticks. Either way, synchronizers merely increase the odds of stabilization and we prefer deterministic correctness guarantees over a probabilistic statement.

In order to demonstrate why, recall that the MTBF is estimated by Equation (3.2). Suppose, we want to synchronize 1 GHz clocks, this yields $F_C = F_D = 10^9\,\mathrm{Hz}$. The values $\tau$ and $T_W$ depend on the technology; SPICE [127] simulations for an ASIC using a 130 nm process yield $\tau = 31.6 \cdot 10^{-12}\,\mathrm{s}$ and $T_W = 8 \cdot 10^{-12}\,\mathrm{s}$ [62]. Assuming that 50 %, 80 %, and 100 % of the clock cycle are available for stabilization yields MTBF estimations of

$$\mathrm{MTBF}_{0.5\,\mathrm{ns}} \approx \frac{e^{15.823}}{8 \cdot 10^6}\,\mathrm{s} \approx 0.9\,\mathrm{s}, \tag{10.1}$$

$$\mathrm{MTBF}_{0.8\,\mathrm{ns}} \approx \frac{e^{25.316}}{8 \cdot 10^6}\,\mathrm{s} \approx 12.3 \cdot 10^3\,\mathrm{s}\ \text{(3.5 hours), and} \tag{10.2}$$

$$\mathrm{MTBF}_{1.0\,\mathrm{ns}} \approx \frac{e^{31.646}}{8 \cdot 10^6}\,\mathrm{s} \approx 6.9 \cdot 10^6\,\mathrm{s}\ \text{(80 days).} \tag{10.3}$$

This means that the MTBF drops to *less than a second per synchronizer* when reserving half the clock cycle for stabilization, rises to about 3.5 hours for the more conservative estimate of 0.8 ns, but never exceeds approximately 80 days, even when leaving no time for computations. Note that these estimates are *per synchronizer* of which $n(n-1)$ are needed, where $n \geq 4$ is required in order to tolerate one fault.

The MTBF is catastrophic for a synchronization delay of 0.8 ns and does not permit the implementation of a mission-critical system, even for a synchronization delay of 1 ns. Hence, we are left with two options. One is to include pipelining in the algorithm of Lundelius Welch and Lynch such that it reacts to measurements only after a sufficient synchronization delay. This means that the difference between clocks indicated by a measurement may grow during the synchronization delay reserved for stabilization of that measurement, hence pipelining reduces the quality of clock synchronization. The alternative is to develop a metastability-containing implementation; we propose one in the following.

Also observe that allotting time $t$ for using synchronizers before beginning the computation imposes a fundamental limit of $1/t$ on the operating frequency. In the light of ever-increasing operating frequencies, we ask: Does the unavoidable presence of metastable upsets when measuring relative timing deviations pose a principal limit on the operating frequency? Below, we argue that this is not the case. We demonstrate the feasibility of a hardware implementation of the Lundelius Welch and Lynch algorithm that does not depend on metastability-free inputs and thus does not suffer from system failures induced by metastable upsets.

## 10.1 Algorithm

The algorithm of Lundelius Welch and Lynch works on $n$ fully connected nodes with a local clock each [106]. Each node generates clock pulses, measures their difference to the other nodes' clock pulses, and adjusts its local clock accordingly. The algorithm tolerates up to $f < n/3$ Byzantine [96] faults.

Our core strategy is a *separation of concerns,* compare Figure 3.1, between the analog and the digital part of the circuit:

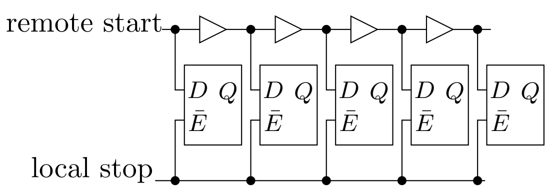


**Figure 10.1:** Tapped delay line TDC. It is read as either $1^k0^{n-k}$ or $1^k\mathrm{M}0^{n-k-1}$, i.e., produces at most one metastable bit and hence has precision-1.

(1) The arrival times of incoming analog clock pulses relative to a node's internal clock are measured using TDCs,

(2) metastability-containing components ensure that the digital part of the circuit handles the potentially partially metastable outcome of the time-to-digital conversion, and

(3) the digital, possibly still partially metastable, signals are converted to analog signals controlling the local clock.

This process provides a *guaranteed end-to-end uncertainty of a single bit* throughout all digital computations, see below, without the hazard of spreading metastability in the control logic in an uncontrolled fashion. Given Marino's result [108], we find it highly surprising that such a thing is possible. The key is that the digital part of the circuit can become metastable, but that metastability is properly contained and ultimately translated into bounded fluctuations in the analog world, not contradicting Marino.

Sophisticated implementations of individual components introduced below have been proposed [26, 62, 100] since the result this chapter is based on [55] first appeared; we point them out below.

We propose a hardware implementation in which each node does the following.

**Step 1: Analog to Digital** First, we step from the analog into the digital world: Delays between remote pulses and the local pulse are measured with TDCs. The measurement can be realized such that at most one of the output bits, accounting for the difference between $x$ and $x + 1$ ticks, becomes metastable; we say such numbers have *precision*-1 and formally define them in Section 10.2.

TDCs can be implemented using tapped delay lines [69, 122, 123], see Figure 10.1: A line of delay elements is tapped in between each two consecutive elements, driving the data input port of initially enabled latches initialized to 0. The rising transition of the remote clock signal fed into the delay line's input passes through the line and sequentially sets the latches to 1; the rising transition of the local clock signal is used to disable all latches at once. After that, the delay line's latches contain the time difference as unary TC. Choosing the propagation delays between the latches larger than their setup/hold times, we ensure that at most one bit is metastable, i.e., that their status is of the form $1^*0^*$ or $1^*\mathrm{M}0^*$. The output is hence a precision-1 TC-encoded time difference.

The recently proposed metastability-containing TDC by Függer et al., building upon our ideas, implements this step and the next [62].

**Step 2: Encoding** We translate the time differences into BRGC, making storage and subsequent components much more efficient. The results are BRGC-encoded time differences with at most one metastable bit of precision-1.

In this step, metastability-containing TC to BRGC conversion is needed and we discuss it in Section 10.3. A more efficient way is to use a metastability-containing TDC which directly produces BRGC of precision-1. As stated above, building on our ideas, such a component has recently been proposed by Függer et al. [62].

**Step 3: Sorting Network** A sorting network selects the $(f+1)$-th and $(n-f)$-th largest remote-to-local clock differences; tolerating $f$ faults requires discarding the top and bottom $f$ values [106].

This requires 2-sort building blocks that pick the minimum and maximum of two precision-1 BRGC-encoded inputs preserving precision-1. We discuss this in Section 10.3; recently, efficient implementations based on our techniques have been presented by Lenzen and Medina [100] and Bund et al. [26], which we improve in Chapter 6.

**Step 4: Decoding and Digital to Analog** The BRGC-encoded $(f+1)$-th and $(n-f)$-th largest remote-to-local clock differences are translated back to TC-encoded numbers. As discussed in Section 10.3, this can be done preserving precision-1, i.e., such that the results are of the form 1*0* or 1*M0*.

This can be used to we step back into the analog world, again without losing precision: The two values are used to control the local clock frequency via a Digitally Controlled Oscillator (DCO). However, the DCO design must be chosen with care. Designs that switch between inverter chains of different length to modify the frequency of a ring oscillator cannot be used, as metastable switches may occur exactly when a pulse passes. Instead, we propose using a ring oscillator whose frequency is controlled by effects such as digitally controlled current sources, capacitor banks, and resistances [39]. While the at most two metastable control bits — at most one in each remote-to-local clock difference — may dynamically change the load of two inverters, this has a limited effect on the overall frequency change and does not lead to glitches within the ring oscillator, because the effect on the clock speed is in the convex hull of the effects of all stabilizations of the digital control parameters.

## 10.2 Encoding and Precision

An appropriate encoding is key to designing metastability-containing components.

**Example 10.1** (Binary). *Suppose a control bit $u$ indicating whether to increase $x = 7$ by 1 is metastable and $x$ is encoded in standard binary. The result must be some resolution of the metastable superposition, see Equation* (4.4)*, of* 00111 *and* 01000*, i.e., anything in* $\mathrm{Res}(00111 \oplus 01000) = \mathrm{Res}(0\mathrm{MMMM})$ *and thus any number $x' \in [16]$, even after stabilization.*

The original uncertainty between 7 and 8 is massively amplified; a good encoding should allow to *contain* the uncertainty imposed by $u = \mathrm{M}$.

A *code* is an injective function $\gamma\colon [n] \to \mathbb{B}^k$ mapping a natural number $x \in [n]$ to its encoded representation. For $y = \gamma(x)$, we define $\gamma^{-1}(y) := x$, and for sets $X$,

$$\gamma(X) := \{\gamma(x) \mid x \in X\} \text{ and} \tag{10.4}$$

$$\gamma^{-1}(X) := \{x \mid \gamma(x) \in X\}. \tag{10.5}$$

We need two encodings: *Thermometer Code (TC)* and *Binary Reflected Gray Code (BRGC).* For the $k$-bit (unary) TC we use

$$\mathrm{un}\colon [k+1] \to \mathbb{B}^k \tag{10.6}$$

$$\mathrm{un}(x) := 0^{k-x}1^x. \tag{10.7}$$

BRGC, compare Figure 10.2(a), is represented by $\mathrm{rg}(x)$ and is much more efficient, using only $\lceil \log_2 n \rceil$ bits. In fact, $\mathrm{rg}\colon [2^k] \to \mathbb{B}^k$ is bijective. We choose un and rg due to the property that in both encodings $\gamma(x)$ and $\gamma(x+1)$ differ in a single bit only. This renders them suitable for metastability-containing operations.

**Example 10.2** (TC and BRGC)**.** *We revisit Example 10.1 with the metastable control bit $u$ indicating whether to increase $x = 7$ by 1.*

**TC** *In 9-bit TC, 7 is encoded as* 001111111 *and* 8 *as* 011111111*, so their metastable superposition resolves to* Res(0M1111111)*, i.e., only to* un(7) *or* un(8)*. Unary encoding, however, is very inefficient regarding the number of bits.*

**BRGC** *Analogously, in 5-bit BRGC, 7 is encoded as* 00100 *and* 8 *as* 01100*, compare Figure 10.2(a), so their metastable superposition resolves to* Res(0M100)*, i.e., only to* rg(7) *or* rg(8)*.*

*Since the original uncertainty is whether or not to increase $x = 7$ by 1, the uncertainty is perfectly contained instead of amplified as with binary code in Example 10.1.*

We formalize the notion of the amount of uncertainty in a partially metastable code word: $x \in \mathbb{B}_{\mathrm{M}}^k$ has *precision-p (w.r.t. the code $\gamma$)* if

$$\max\left\{y - \bar{y} \mid y, \bar{y} \in \gamma^{-1}(\mathrm{Res}(x))\right\} \le p, \tag{10.8}$$

i.e., if the largest possible difference between resolutions of $x$ is bounded by $p$.

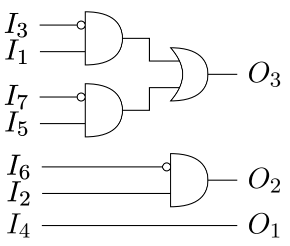
Note that the components presented below make heavy use of BRGC. This makes them more involved, but they are exponentially more efficient than their TC counterparts in terms of memory and avoid the amplification of uncertainties incurred by standard binary encoding. As a matter of fact, recently proposed efficient implementations for metastability-containing sorting networks [26, 100] and metastability-containing TDCs [62] use BRGC.

## 10.3 Digital Components

In the following, we show that all metastability-containing components required for the clock synchronization algorithm outlined in Section 10.1 exist. As motivated above, the components have to maintain meaningful outputs in face of limited metastability; more precisely, we deal with the abovementioned precision-1 output of TDCs.

| dec | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 |
| 4 | 1 | 1 | 0 |
| 5 | 1 | 1 | 1 |
| 6 | 1 | 0 | 1 |
| 7 | 1 | 0 | 0 |

**(a)** 3-bit BRGC.



**(b)** Transformation circuit.

**Figure 10.2:** Efficient TC-to-BRGC conversion.

**TC-to-BRGC Converter** Figure 10.2(b) depicts a circuit that translates 7-bit TC into 3-bit BRGC. It generalizes to $n$-bit inputs, with a depth of $\lceil \log_2 n \rceil$. While such translation circuits are well-known, it is important to check that the given circuit fulfills the required property of preserving precision-1: This holds as each input bit influences exactly one output bit and, due to the structure of BRGC, this bit makes exactly the difference between $\mathrm{rg}(x)$ and $\mathrm{rg}(x+1)$ given a TC-encoded input of $1^x\mathrm{M}0^{7-x-1}$.

**Sorting Networks** It is well-known that sorting networks can be efficiently composed from 2-sort building blocks [3, 11], which map $(x, y)$ to $(\min\{x, y\}, \max\{x, y\})$. We show that max (and analogously min) of two precision-1 $k$-bit BRGC numbers is implementable with combinational logic, i.e., without masking registers, such that each output has precision-1. Observe that this is straightforward for TC-encoded inputs with bit-wise And and Or for min and max, respectively. However, this is possible for BRGC inputs as well.

**Lemma 10.3.** *Define* $\max_{BRGC}\colon \mathbb{B}^k \times \mathbb{B}^k \to \mathbb{B}^k$ *as*

$$\max_{BRGC}(x, y) := \mathrm{rg}\left(\max\left\{\mathrm{rg}^{-1}(x), \mathrm{rg}^{-1}(y)\right\}\right). \tag{10.9}$$

*Then* $[\max_{BRGC}]_\mathrm{M} \in \mathrm{Fun}_S$ *and it determines precision-*1 *output from precision-*1 *inputs* $x$ *and* $y$*.*

*Proof.* Since $x$ and $y$ have precision-1, $\mathrm{rg}^{-1}(\mathrm{Res}(x)) \subseteq \{a, a+1\}$ for some $a \in [2^k - 1]$ (analogously for $y$ w.r.t. some $b \in [2^k - 1]$). W.l.o.g. assume $a \geq b$, i.e., for all possible resolutions of $x$ and $y$, the circuit must output $\mathrm{rg}(a)$ or $\mathrm{rg}(a+1)$. By Definition 9.3 and the fact that $\mathrm{rg}(a)$ and $\mathrm{rg}(a+1)$ differ in a single bit only, $[\max_\mathrm{BRGC}]_\mathrm{M}(x, y)$ has at most one metastable bit and precision-1. □

An analogous argument holds for

$$\min_\mathrm{BRGC}(x, y) := \mathrm{rg}\left(\min\left\{\mathrm{rg}^{-1}(x), \mathrm{rg}^{-1}(y)\right\}\right). \tag{10.10}$$

Observe that sorting precision-1 BRGC numbers may lead to the following effect: Consider $\mathrm{rg}(6) \oplus \mathrm{rg}(7) = 10\mathrm{M}$, the metastable superposition of 6 and 7 in 3-bit BRGC, compare Figure 10.2(a). Then

$$[\max_{\mathrm{BRGC}}]_{\mathrm{M}}(10\mathrm{M}, 10\mathrm{M}) = \mathrm{Res}_{\mathrm{M}}(10\mathrm{M}) = [\min_{\mathrm{BRGC}}]_{\mathrm{M}}(10\mathrm{M}, 10\mathrm{M}). \tag{10.11}$$

Hence, it is possible that after stabilization $100 = \mathrm{rg}(7)$ is ordered before $101 = \mathrm{rg}(6)$. By definition of the metastable closure and due to using precision-1 BRGC input, however, the flipped order can only occur between numbers that differ by at most 1, i.e., the error is bounded by the initial uncertainty.

Based on our proposed 2-sort building blocks, Lenzen and Medina [100] and Bund et al. [26] present implementations using $\mathrm{O}(k^2)$ and $\mathrm{O}(k \log k)$ gates, respectively.

Metastability-containing sorting networks open up another option. If we insist on using synchronizers, we can still deploy them *after* sorting and selecting the $(f+1)$-th and $(n-f)$-th largest remote-to-local clock differences. While this does not remove the synchronization delay, it reduces the total number of synchronizers from $n(n-1)$ to $2n$, i.e., from $\mathrm{O}(n^2)$ to $\mathrm{O}(n)$. This trade-off, however, may only become interesting for large values of $n$ and low-overhead precision-1 BRGC comparators.

**BRGC-to-TC Converter** A BRGC-encoded number of precision-1 has at most one metastable bit: For any up-count from an encoding of $x \in [2^k - 1]$ to $x + 1$ a single bit changes, which thus can become metastable if it has precision-1. It is possible to preserve this guarantee when converting to TC.

**Lemma 10.4.** *Define* $\mathrm{rg2un}\colon \mathbb{B}^k \to \mathbb{B}^{(2^k-1)}$ *as*

$$\mathrm{rg2un}(x) := \mathrm{un}\left(\mathrm{rg}^{-1}(x)\right). \tag{10.12}$$

*Then* $[\mathrm{rg2un}]_{\mathrm{M}} \in \mathrm{Fun}_{\mathcal{S}}$ *converts its parameter to TC, preserving precision*-1.

*Proof.* If $x$ has precision-1, then $\mathrm{rg}^{-1}(\mathrm{Res}(x)) \subseteq \{a, a+1\}$ for some $a \in [2^k - 1]$. Hence, $\mathrm{un}(a)$ and $\mathrm{un}(a+1)$ differ exactly in the $(a+1)$-th bit, proving the claim. □

# CHAPTER 11
# Conclusion

No digital circuit can reliably avoid, detect, or resolve metastable upsets [108]. While the standard approach is to use synchronizers to trade time for an increased MTBF [13, 14, 15, 67, 88, 89], we propose a fundamentally different method: It is possible to design efficient digital circuits that tolerate a certain degree of metastability in the input. This technique has critical advantages:

(1) Where synchronizers decrease the odds of failure, our techniques provide deterministic guarantees. A synchronizer may or may not stabilize in the allotted time frame. Our model, on the other hand, guarantees to return one of a specific set of known values — e.g., the metastable closure, but this depends on the application — without relying on probabilities.

(2) Our approach saves time and, by eliminating synchronizers, removes one fundamental limitation on the operating frequency. If the required functions can be implemented in a metastability-containing way, there is no need to reserve time for synchronization before starting the computation.

(3) If metastability needs to be resolved eventually, one can still save time by allowing for stabilization *during* the metastability-containing computations.

In light of these properties, we expect our techniques to prove useful for a variety of applications, in particular in time- and mission-critical scenarios.

Our argumentation is based on a model developed in Chapter 4. It captures clocked and combinational circuits and models the spread of metastability in a worst-case fashion. In the absence of metastability, it behaves like a regular circuit model. We check the merits of our model by showing in Chapter 5 that it reproduces the known impossibility of avoiding, detecting, or resolving metastability. Further, we demonstrate in Chapter 6 that the explanatory power of our model extends to the transistor level by developing CMOS implementations of CMUXes.

We show in Chapters 7 and 8 that circuits with simple registers can be unrolled and are computationally equivalent to combinational circuits. In this case, our model can be simplified to Kleene logic in combinational logic DAGs, making reasoning about circuits much more convenient than in the general case. The presence of masking registers, however, strictly increases the computational power with each clock cycle; we are able to establish a strict inclusion result. Furthermore, we fully classify all functions implementable in combinational and simple circuits in Chapter 9.

As a consequence of our techniques, we establish the implementability of the fault-tolerant clock synchronization algorithm by Lundelius Welch and Lynch [106] with a deterministic correctness guarantee, despite the unavoidable presence of metastable upsets, in Chapter 10.

**Recent Developments** A range of promising results has been obtained based on our work: metastability-containing sorting networks [26, 100], a TDC that directly produces precision-1 BRGC [62], and metastability-aware network-on-chip routers [128]. Together with verification in simulations [26, 60, 128], we believe this to strengthen the case for the explanatory power of our model.

**Future Work** We focus on computability under metastable inputs. There are many open questions regarding circuit complexity in our model of computation.

It is of interest to reduce the gate complexity and latency of concrete circuits, as well as to determine the complexity overhead of metastability-containment in general. In particular, the overhead of implementing the metastable closure $[f]_{\mathrm{M}}$ as compared to an implementation of $f$ that is oblivious to metastability — and if there has to be an overhead at all — is an open question, in general as well as for particular functions $f$, see Section 9.3.

Masking registers are computationally strictly more powerful than simple registers (Theorem 8.3). An open question is which metastability-containing circuits benefit from masking registers and to find further examples separating $\mathrm{Fun}_M^r$ from $\mathrm{Fun}_M^{r+1}$.

Our model does not capture clock gating, i.e., non-input registers are overwritten in every clock cycle. Hence, storing intermediate results in masking registers is pointless: Taking advantage of at most one read from an input masking-register becoming metastable does not apply to results of intermediate computations. It is an open problem whether this makes a difference in terms of computability.

# PART II

## Parallel Distance Problems

*"Confusion is the only thing that grows when shared."*

A confused meerkat

This part is the result of close collaboration with Christoph Lenzen. The text is based on an article that is under submission to the *Journal of the ACM (JACM)* as of August 2016, an extended abstract of which appeared in the *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2016),* pages 455–466, 2016 [61].

# CHAPTER 12

# Introduction

Many problems in computer science are intrinsically linked to distances in graphs. This certainly is due to the fact that graphs are an extremely useful tool for modeling relationships between discrete entities. Hence, it is not surprising that some of the most classic algorithms determine distances in graphs, e.g., Dijkstra's algorithm [41] and the Moore-Bellman-Ford (MBF) algorithm [16, 54, 116]. But distance problems are by no means a closed book; recent literature offers plenty of results on this front. We cannot hope to give an exhaustive list of related work, but would like to mention distance approximations [20, 34, 48, 73, 91, 101, 103, 135], spanners [10], hop sets [34, 48, 73], algebraic distance computations [5, 35, 115, 135], metric tree embeddings [6, 8, 9, 20, 21, 50, 65, 81, 110], and distributed algorithms [10, 65, 73, 81, 101, 102, 103, 104, 107, 119].

Another highly relevant topic are parallel computations [10, 19, 21, 22, 33, 34, 48, 77, 91, 105, 107, 113]. Gigantic data centers, the distributed storage and processing of information, and large-scale computer networks are commonplace. Given the relevance of graph problems and that they grow more and more complex, it is highly relevant to harness the power of parallel machines in this context, as pointed out by, e.g., Lumsdaine et al. [105].

Our goal are polylogarithmic *depth* (parallel time) algorithms of little *work* (total number of operations). The work can be thought of as the time required by a sequential machine and the depth as a lower bound on the time required by a parallel machine with an arbitrary number of processors and without synchronization overhead. Ideally, an algorithm's work is close to the time of a fast sequential algorithm for the same problem.

We study parallel computations of distance problems in graphs; our focus are distance approximations that preserve the triangle inequality. To see the relevance in this, let $G = (V, E, \omega)$ be a graph with $n$ vertices and $m$ edges and recall that exact distances in $G$, denoted by $\operatorname{dist}(\cdot,\cdot,G)\colon V \times V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$,[1] fulfill the *triangle inequality:*

$$\forall u, v, w \in V\colon \quad \operatorname{dist}(u, w, G) \leq \operatorname{dist}(u, v, G) + \operatorname{dist}(v, w, G), \tag{12.1}$$

i.e., the "direct route" from $u$ to $w$ is never heavier than any "detour" visiting $u$, $v$, and $w$ in order. The triangle inequality is one of the properties required from a metric.

Exact distances automatically fulfill the triangle inequality. In order to obtain efficient parallel algorithms, however, we often resort to approximate distances. The problem is that if $d\colon V \times V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is an $\alpha$-approximation of $\operatorname{dist}(\cdot,\cdot,G)$, the triangle inequality does not necessarily hold in $d$. This is irrelevant for some problems — e.g., finding an approximately shortest path between two vertices — but it is highly relevant for others. As an example, the probabilistic metric tree embedding of Fakcharoenphol, Rao, and Talwar (FRT) relies on the subtractive form of the triangle inequality [50].

[1]Below, we use the notation introduced in Section 12.3 without explicit forward references.

Hence, we study approximate, parallel, polylogarithmic-depth, work-efficient algorithms that either explicitly compute or implicitly work on *approximate metrics,* i.e., obey the triangle inequality. We list our contributions below, however, one key result are algorithms that w.h.p. determine a $(1+\mathrm{o}(1))$- and $\mathrm{O}(1)$-approximate metric of $\operatorname{dist}(\cdot,\cdot,G)$ using $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(n(m+n^{1+\varepsilon}))$ and $\tilde{\mathrm{O}}(n^{2+\varepsilon})$ work, respectively. Observe that the latter is close to the trivial lower bound of $\Omega(n^2)$ work for writing the result. Another key result is that we can w.h.p. sample a probabilistic FRT-style tree embedding of $G$ that has an expected stretch of $\mathrm{O}(\log n)$ using $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work. The work can be reduced to $\tilde{\mathrm{O}}(m+n^{1+\varepsilon})$ at the expense of increasing the expected stretch to $\mathrm{O}(\varepsilon^{-1}\log n)$. In both cases, the work is close to $\mathrm{O}(m\log n)$, the state of the art in the sequential setting [20].

**MBF-like Algorithms** In order to obtain algorithms that use polylogarithmic depth while at the same time being work-efficient and guaranteeing consistency with the triangle inequality, we resort to a virtual graph $H$. It has a Shortest-Path Diameter (SPD) of $\mathrm{O}(\log^2 n)$, meaning that we can find shortest paths with at most $\operatorname{SPD}(H) \in \mathrm{O}(\log^2 n)$ hops between each pair of nodes. This resolves the issue with the triangle inequality — we compute exact distances in $H$ which automatically form a metric — and maintains the possibility of polylogarithmic depth, as $\mathrm{O}(\log^2 n)$ iterations suffice. Assuming we may add an appropriate hop set [34] to $G$, $\operatorname{dist}(\cdot,\cdot,H)$ is a $(1+\mathrm{o}(1))$-approximation of $\operatorname{dist}(\cdot,\cdot,G)$, meaning that the computed distances are essentially as good as those in $G$.

Unfortunately, $H$ is a complete graph. Hence, explicitly computing the adjacency lists of $H$, or even performing a single iteration in $H$, incurs $\Omega(n^2)$ work; this defeats the hope of near-linear work in the number of edges of $G$.

Fortunately, we can simulate the iterations of an algorithm in $H$ using iterations in $G$ and polylogarithmic overhead in both work and depth. The simulation requires some structural properties of the algorithm, but generally works for algorithms that "behave like the MBF algorithm." We formalize this notion, propose the class of *Moore-Bellman-Ford-like (MBF-like) algorithms,* and show that the simulation argument holds for them.

Our classification of MBF-like algorithms provides a new perspective on a classical scheme of spreading information through a graph that is a key ingredient to many algorithms [16, 54, 65, 73, 81, 101, 102, 103, 104, 116]. Further, it captures a wide range of known algorithms, including but not limited to source detection, Single-Source Shortest Paths (SSSP), $k$-Source Shortest Paths ($k$-SSP), All-Pairs Shortest Paths (APSP) and Multi-Source Shortest Paths (MSSP) over the min-plus semiring, Single-Source Widest Paths (SSWP), All-Pairs Widest Paths (APWP) and Multi-Source Widest Paths (MSWP) over the max-min semiring, the more involved $k$-Shortest Distance Problem ($k$-SDP) and $k$-Distinct-Shortest Distance Problem ($k$-DSDP) over the all-paths semiring, and connectivity over the Boolean semiring. As the above list is by no means complete, we conclude that our classification is rather flexible and consider it of independent interest.

The basic observation is that in an MBF-like algorithm each node in each iteration (1) *propagates* the information obtained so far to all its neighbors, (2) then *aggregates* the received information, and (3) optionally *filters* out irrelevant parts. As an example, consider $k$-SSP, where we determine, for each node, the $k$ nodes closest to it. Each node stores node–distance pairs (initially only themselves at distance 0) and then iterates the following steps: (1) propagate the node–distance pairs to the neighbors, uniformly increasing distances by the corresponding edge weight, (2) aggregate the received values by picking the minimum distance for each node, and (3) discard node–distance pairs for which there are $k$ closer alternatives.

An MBF-like algorithm has three key components: a semiring $\mathcal{S}$, a semimodule $\mathcal{M} = (M, \oplus, \odot)$ over $\mathcal{S}$, and a filter $r \colon \mathcal{M} \to \mathcal{M}$. The semiring captures how distances and edge weights interact in the underlying graph. A standard choice is the min-plus semiring that is a common tool for distance problems [5, 35, 115, 135]. We use the semimodule $\mathcal{M}$ as a "data structure" for collecting information that can be accumulated during the algorithm. $\mathcal{M}$ might be a single distance when solving SSSP or a vector of $n$ distances for APSP. Propagation corresponds to $\odot$ and aggregation to $\oplus$.

The filter is a projection that discards information which is irrelevant for the algorithm at hand. Roughly speaking, in an $h$-iteration MBF-like algorithm, each node determines its part of the output based on its $h$-hop distances to all other nodes. For efficiency reasons, however, various algorithms [65, 73, 81, 101, 102, 103, 104, 115] compute only a subset of these distances. The role of the filter is to remove the irrelevant values to allow for better efficiency. The core feature of an MBF-like algorithm is that filtering is "compatible" with propagation and aggregation: If a node discards information and then propagates it, the discarded parts must be "uninteresting" at the receiving node as well. We model this using a *congruence relation* on the node states; filters pick a suitable (efficiently encodable) representative of the node state's equivalence class.

We hope that relatively few standard choices for $\mathcal{S}$ and $\mathcal{M}$ can be reused for many algorithms, leaving it up to the filter to implement the algorithm-specific customization. Hence, we hope that the framework of MBF-like algorithms, besides capturing many algorithms, easily extends to further algorithms. This is different from Mohri's framework which requires a custom semiring for each algorithm [115].

**Our Approach** APSP and MSSP determine a metric and a submetric, respectively, and collecting Least Element (LE) lists [33] readily allows to sample an FRT tree [21, 81], the abovementioned tree embedding of expected logarithmic stretch. These algorithms are MBF-like, hence we may simulate them in $H$ to obtain good approximations of what these algorithms would determine in $G$. Observe that running any of these algorithms in $G$ to determine exact distances — in order to respect the triangle inequality — takes $\mathrm{SPD}(G)$ iterations, where $\mathrm{SPD}(G) = n - 1$ is possible. This corresponds to linear depth and hence is not an option in our setting. We sample an FRT tree (alternatively, determine a metric or a submetric) using the following sequence of embeddings. Starting with the graph $G$, we

(1) first obtain $G'$ by adding a $(d, \hat{\varepsilon})$-hop set, where $d, \hat{\varepsilon}^{-1} \in \operatorname{polylog} n$ to $G$ [34], then

(2) metrically embed $G'$ into $H$ (implicitly), and

(3) embed $H$ into an FRT tree (alternatively, determine a metric or submetric).

Is the second step required, i.e., is there no hop-set algorithm that ensures that $d$-hop distances satisfy the triangle inequality? Unfortunately, hop sets do not solve the problem.

First recall that a $(d, \hat{\varepsilon})$-hop set is a set of edges that, when added to the graph, ensures that $d$-hop distances in $G'$ approximate $\operatorname{dist}(\cdot, \cdot, G)$, i.e., ensure that

$$\forall v, w \in V\colon \quad \operatorname{dist}(v, w, G) \leq \operatorname{dist}^d(v, w, G') \leq (1 + \hat{\varepsilon}) \operatorname{dist}(v, w, G), \tag{12.2}$$

where $G'$ is $G$ augmented with the hop-set edges [34]. Put differently, hop sets guarantee that $d$-hop distances approximate distances. However, any hop set that fulfills the triangle inequality on $d$-hop distances has to reduce the SPD to at most $d$, i.e., yield exact distances, which is a much harder problem:

**Observation 12.1.** *Let $G$ be a graph that contains a $(d, \hat{\varepsilon})$-hop set. If $\operatorname{dist}^d(\cdot, \cdot, G)$ is a metric, then $\operatorname{dist}^d(\cdot, \cdot, G) = \operatorname{dist}(\cdot, \cdot, G)$, i.e., $\operatorname{SPD}(G) \leq d$.*

*Proof.* Let $\pi$ be a shortest $u$-$v$-path in $G$. Since $\operatorname{dist}^d(\cdot, \cdot, G)$ fulfills the triangle inequality,

$$\operatorname{dist}^d(u, v, G) \leq \sum_{\{u_1, u_2\} \in \pi} \operatorname{dist}^d(u_1, u_2, G) \leq \sum_{\{u_1, u_2\} \in \pi} \omega(u_1, u_2) = \operatorname{dist}(u, v, G) \tag{12.3}$$

holds and the claim follows due to $\operatorname{dist}(u, v, G) \leq \operatorname{dist}^d(u, v, G)$. □

This is why it is not an option to determine LE lists based on $d$-hop distances in $G'$, which would be a straightforward adjustment of the algorithm of Khan et al. [81]. As pointed out by Ghaffari and Lenzen [65], the FRT construction depends on the triangle inequality [50], but $\operatorname{dist}^d(\cdot, \cdot, G')$ does not guarantee it, not even in the presence of a $(d, \hat{\varepsilon})$-hop set. We overcome this issue by embedding $G'$ in $H$. Where hop sets preserve distances *exactly* and ensure the existence of *approximately* shortest paths with few hops, we preserve distances *approximately* but guarantee to obtain them *exactly* with few hops.

**Metric Tree Embeddings** In many graph problems the objective is closely related to distances in the graph. Prominent examples are shortest path problems, Minimum Spanning Trees (MSTs), a plethora of Steiner-type problems [71], the Traveling Salesman Problem (TSP), and many more. If approximation is viable or mandatory, a successful strategy is to approximate the distance structure of the weighted graph $G$ by a simpler graph $G'$, where "simpler" can mean fewer edges, smaller degrees, being from a specific family of graphs, or any other constraint making the considered problem easier to solve. One then proceeds to solve a related instance of the problem in $G'$ and maps the solution back to $G$, yielding an approximate solution of the original instance. Naturally, this requires a mapping of bounded impact on the objective value.

*Metric embeddings* map the graph $G = (V, E, \omega)$ to a graph $G' = (V', E', \omega')$, such that $V \subseteq V'$ and

$$\forall v, w \in V\colon \quad \operatorname{dist}(v, w, G) \leq \operatorname{dist}(v, w, G') \leq \alpha \operatorname{dist}(v, w, G) \tag{12.4}$$

for some $\alpha \in \mathbb{R}_{\geq 1}$ referred to as *stretch.* The abovementioned hop sets are an example for a metric embedding. An especially convenient class of metric embeddings are *metric tree embeddings,* where $G'$ is a tree, plainly because very few problems are hard to solve in tree instances. The utility of tree embeddings originates in the fact that, despite their extremely simple topology, it is possible to randomly construct an embedding of any graph $G$ into a tree $T$ so that the *expected stretch*

$$\alpha = \max_{v \neq w \in V} \frac{\mathbb{E}_T\left[\operatorname{dist}(v,w,T)\right]}{\operatorname{dist}(v,w,G)} \tag{12.5}$$

satisfies $\alpha \in \mathrm{O}(\log n)$ [50]. By linearity of expectation, this ensures an expected approximation ratio of $\mathrm{O}(\log n)$ for many problems.

A substantial advantage of tree embeddings lies in the simplicity of applying the machinery: Translating the instance in $G$ to one in $T$, solving the instance in $T$, and translating the solution back tends to be extremely efficient and highly parallelizable. Hence, a low-depth small-work parallel algorithm for embedding weighted graphs into trees in the vein of FRT gives rise to fast and efficient parallel approximations for many graph problems. Unfortunately, the state of the art regarding polylog $n$ depth uses $\Omega(n^2)$ work and requires constant-time query access to $\operatorname{dist}(\cdot,\cdot,G)$ [21], whereas the best sequential algorithm only uses $\mathrm{O}(m \log n)$ time w.h.p. [20].

## 12.1 Our Contribution

Our contributions are organized in techniques and their application. We establish the required techniques in Chapters 13–15:

(1) Our key tool is the algebraic classification of MBF-like algorithms described in Chapter 13. As our framework subsumes a large class of known algorithms and explains them from a different perspective, we consider it to be of independent interest.

(2) We demonstrate that our framework of MBF-like algorithms captures a wide range of algorithms using numerous examples in Chapter 14.

(3) Chapter 15 proposes a sampling technique for embedding a graph $G$ in which $d$-hop distances $(1+\hat{\varepsilon})$-approximate exact distances — a graph that contains a $(d,\hat{\varepsilon})$-hop set — into a complete graph $H$, where $H$ has an SPD of $\mathrm{O}(\log^2 n)$ and preserves $G$-distances $(1+\hat{\varepsilon})^{\mathrm{O}(\log n)}$-approximately. The polylogarithmic SPD of $H$ is key to achieving polylogarithmic-depth algorithms that obey the triangle inequality. Unfortunately, we cannot use $H$ directly, because it is a complete graph and hence imposes $\Omega(n^2)$ work per iteration. We can, however, simulate an iteration of an MBF-like algorithm $\mathcal{A}$ in $H$ using only $\tilde{\mathrm{O}}(d)$ iterations $G$. Hence, we can simulate all $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ iterations of $\mathcal{A}$ in $H$ using only polylogarithmic overhead in depth and work w.r.t. a single iteration of $\mathcal{A}$ in $G$. We formulate this as an oracle theorem: The oracle is provided with $G$ and $\mathcal{A}$, and answers with the result of running $\mathcal{A}$ in $H$. Together with an appropriate hop-set algorithm, i.e., one that guarantees $d \in$ polylog $n$, this allows for algorithms of polylog $n$ depth and $\tilde{\mathrm{O}}(m)$ work.

We apply the above techniques to establish the results in Chapters 16–19. The strongest result is our algorithm that w.h.p. samples an FRT tree using polylog $n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work in Chapter 17.

(4) A first consequence of our techniques is that we can query the oracle with APSP to determine $\operatorname{dist}(\cdot,\cdot,H)$, i.e., to efficiently calculate a metric that approximates $\operatorname{dist}(\cdot,\cdot,G)$. For a constant $0<\varepsilon\leq\frac{1}{2}$, we w.h.p. obtain a $(1+\mathrm{o}(1))$-approximate metric of $\operatorname{dist}(\cdot,\cdot,G)$ using $\tilde{\mathrm{O}}(n(m+n^{1+\varepsilon}))$ work and $\operatorname{polylog} n$ depth. Alternatively, we can apply the spanner construction of Baswana and Sen [10] as a preprocessing step to w.h.p. obtain an $\mathrm{O}(1)$-approximation using $\tilde{\mathrm{O}}(n^{2+\varepsilon})$ work and $\operatorname{polylog} n$ depth. Both results generalize to submetrics by using MSSP instead of APSP. We discuss this in Chapter 16.

(5) In Chapter 17, we show that for any constant $0<\varepsilon\leq\frac{1}{2}$, there is a randomized parallel algorithm of $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work that computes a metric tree embedding of expected stretch $\mathrm{O}(\log n)$ w.h.p. This is possible because determining LE lists [33] is an efficient MBF-like algorithm, which in turn allows us to sample an FRT tree on $H$ using the oracle. Applying the spanner construction of Baswana and Sen [10] as a preprocessing step, the work can be reduced to $\tilde{\mathrm{O}}(m+n^{1+\varepsilon})$ at the expense of stretch $\mathrm{O}(\varepsilon^{-1}\log n)$. We improve upon the state of the art by Blelloch et al., who require $\mathrm{O}(n^2)$ work for sampling the FRT tree and require constant-time query access to $\operatorname{dist}(\cdot,\cdot,G)$ to do so [21]. Further observe that our work almost matches the state of the art sequential-time algorithm of Blelloch et al. which w.h.p. requires $\mathrm{O}(m\log n)$ time [20].

(6) Our techniques allow to improve over previous algorithms computing tree embeddings in the CONGEST model, a standard model of computation for distributed algorithms. We improve upon the algorithm by Ghaffari and Lenzen, which computes a tree embedding of expected stretch $\mathrm{O}(\varepsilon^{-1}\log n)$ in $\tilde{\mathrm{O}}(\min\{n^{1/2+\varepsilon},\mathrm{SPD}(G)\}+\mathrm{D}(G))$ rounds [65], by reducing the expected stretch to $\mathrm{O}(\log n)$ and the round complexity to $\min\{\tilde{\mathrm{O}}((n^{1/2}+\mathrm{D}(G))n^{\mathrm{o}(1)}),\tilde{\mathrm{O}}(\mathrm{SPD}(G))\}$. This is detailed in Chapter 18.

(7) We further illustrate the utility of our results by providing efficient approximation algorithms for the $k$-median and buy-at-bulk network design problems. Blelloch et al. propose polylogarithmic-depth parallel algorithms based on FRT embeddings for these problems that either require constant-time query access to $\operatorname{dist}(\cdot,\cdot,G)$ as input or first determine it by solving APSP [21]. We remove this constraint, resulting in more general and more work-efficient algorithms. The details are discussed in Chapter 19.

Chapter 20 concludes this part.

## 12.2 Related Work

We confine the discussion to undirected graphs.

**Classical Distance Computations** The earliest — and possibly most basic — algorithms for SSSP computations are Dijkstra's algorithm [41] and the Moore-Bellman-Ford (MBF) algorithm [16, 54, 116]. From the perspective of parallel algorithms, Dijkstra's algorithm performs excellent in terms of work, requiring $\tilde{\mathrm{O}}(m)$ computational steps, but suffers from being inherently sequential, processing one vertex at a time.

**Algebraic Distance Computations** Distance computations can be performed by multiplication with the weighted adjacency matrix $A$ over the min-plus, a.k.a. tropical, semiring $\mathcal{S}_{\min,+} = (\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +)$ [5, 35, 115, 135]. The MBF algorithm can be interpreted as a fixed-point iteration $x^{(i+1)} := Ax^{(i)}$, where addition and multiplication happen in $\mathcal{S}_{\min,+}$, i.e., are replaced by min and $+$, respectively. This is a well-established tool for distance computations and from this point of view, $\mathrm{SPD}(G)$ is the number of iterations until a fixed point is reached. The MBF algorithm thus permits depth $\tilde{\mathrm{O}}(\mathrm{SPD}(G))$ and work $\tilde{\mathrm{O}}(m\,\mathrm{SPD}(G))$, where small $\mathrm{SPD}(G)$ are possible.

One may overcome the issue of large depth entirely by performing the fixed-point iteration on $A$, by setting $A^{(0)} := A$ and iterating $A^{(i+1)} := A^{(i)}A^{(i)}$; this guarantees that a fixed point is reached after $\lceil \log_2 \mathrm{SPD}(G) \rceil \leq \lceil \log_2(n-1) \rceil$ iterations [35]. The final matrix then has as entries exactly the pairwise node distances, and the computation has polylogarithmic depth. This comes at the cost of $\Omega(n^3)$ work, even if $m \ll n^2$. If the graph is dense, however, it is as work-efficient as solving APSP using $n$ SSSP instances of Dijkstra's algorithm — of $\mathrm{O}(n \log n + m)$ work each — without incurring depth $\Omega(n)$.

Mohri solved various shortest-distance problems using the $\mathcal{S}_{\min,+}$ semiring and variants thereof [115]. This framework is quite general, but our approach is different in key aspects:

(1) Mohri's algorithm can be interpreted as a generalization of Dijkstra's algorithm [41], because it maintains a global queue and, in each iteration, applies a relaxation technique to the dequeued element and its neighbors. Like Dijkstra's algorithm, this strategy is inherently sequential. We do not need a global queue and, to the best of our knowledge, are the first to present a general algebraic framework for distance computations that exploits the implicit parallelism of the MBF algorithm.

(2) Mohri uses an individual semiring for each problem and solves it by a general algorithm. Our approach, on the other hand, is more generic as well as easier to use: We use off-the-shelf semirings like $\mathcal{S}_{\min,+}$ and combine them with appropriate, usually standard, semimodules that carry enough problem-specific information. The problem-specific customization happens in the filter that discards information irrelevant for the problem at hand. We demonstrate the modularity and flexibility of the approach by various examples in Chapter 14, which cover a large variety of distance problems, including Mohri's.

(3) In our framework, node states are semimodule elements and edge weights are semiring elements; hence, there is no multiplication of node states. Mohri's approach, however, does not make this distinction and hence requires an artificial multiplication of node states.

(4) In Mohri's approach, choosing the global queuing strategy is not only an integral part of an algorithm, but also simplifies the construction of the underlying semirings, as one may rule that elements are processed in a convenient order. Our framework is flexible enough to achieve counterparts even of Mohri's more involved results without such assumptions; concretely, we propose a suitable semiring for solving $k$-SDP and $k$-DSDP in Chapter 14.

**Approximate Distance Computations** As metric embeddings reproduce distances only approximately, we may base them on approximate distance computations in the original graph. Using rounding techniques to reduce the matrix–matrix product over $\mathcal{S}_{\min,+}$ to one over $\mathbb{R}$, Zwick is able to use fast matrix multiplication to speed up the aforementioned fixed-point iteration $A^{(i+1)} := A^{(i)}A^{(i)}$ [135]. This reduces the work to $\tilde{\mathrm{O}}(\varepsilon^{-1}n^{\omega})$ at the expense of only $(1+\varepsilon)$-approximating distances, yielding $\tilde{\mathrm{O}}(n^{\omega})$ work for a $(1+\mathrm{o}(1))$-approximation for $\varepsilon^{-1} = \log n$. Here, $\omega$ denotes the fast matrix-multiplication exponent. However, even if the conjecture that $\omega = 2$ holds true, this technique must result in $\Omega(n^2)$ work, simply because $\Omega(n^2)$ pairwise distances are computed.

Regarding SSSP, there was no work-efficient low-depth parallel algorithm for a long time, even when allowing approximation: Matrix–matrix multiplication was inefficient in terms of work, while sequentially exploring (shortest) paths resulted in depth $\Omega(\mathrm{SPD}(G))$. Klein and Subramanian [91] showed that depth $\tilde{\mathrm{O}}(\sqrt{n})$ can be achieved with $\tilde{\mathrm{O}}(m\sqrt{n})$ work, beating the $\Omega(n^2)$ work barrier with sublinear depth in sparse graphs.

In a seminal paper, Cohen [34] proved that SSSP can be $(1+\mathrm{o}(1))$-approximated at depth $\operatorname{polylog} n$ and near-optimal $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work, for an arbitrary constant choice of $0 < \varepsilon \leq \frac{1}{2}$. Her approach is based on constructing the aforementioned $(d, \hat{\varepsilon})$-hop set, where $d, \hat{\varepsilon}^{-1} \in \operatorname{polylog} n$, and then computing $d$-hop distances. Henzinger et al. [73] determine an $(n^{\mathrm{o}(1)}, \mathrm{o}(1))$-hop set using $(n^{1/2} + \mathrm{D}(G))n^{\mathrm{o}(1)}$ rounds in the CONGEST model; their construction, however, does not yield a parallel algorithm of $\operatorname{polylog} n$ depth. Recently, Elkin and Neiman obtained hop sets with improved trade-offs [48], both for the parallel setting and the CONGEST model. Our embedding technique is formulated independently from the underlying hop-set construction, the performance of which is reflected in the depth and work bounds of our algorithms. Elkin and Neiman, however, do not enable us to simultaneously achieve work $m^{1+\mathrm{o}(1)}$ and depth $\operatorname{polylog} n$, while maintaining our approximation guarantees.

**$\Delta$-Stepping** Other than resorting to approximation algorithms, authors have worked on parallelizing exact SSSP computations based on Dijkstra's algorithm [41], processing multiple elements of the queue used in Dijkstra's algorithm in parallel. The most prominent example for this is the $\Delta$-stepping algorithm by Meyer and Sanders [113]. It maintains buckets of nodes that have a tentative distance between $i\Delta$ and $(i+1)\Delta$ from the source node $s$, where $\Delta \in \mathbb{R}_{>0}$ is a parameter of the algorithm, and processes the nodes of the non-empty bucket closest to $s$ in parallel. The performance depends on $\Delta$, the maximum node degree and the maximum weight of a shortest $s$-$v$ path, and can be impeded by reinsertions in the active bucket. However, $\Delta$-stepping performs well on certain random graphs and road networks [113]. Regarding practical implementations, Lumsdaine et al. obtained promising results from several $\Delta$-stepping implementations [105].

**Metric Tree Embeddings** When metrically embedding into a tree, it is, in general, impossible to guarantee a small stretch. For instance, when the graph is a cycle with unit edge weights, it is impossible to embed it into a tree without having at least one edge with stretch $\Omega(n)$ [8]. When discarding an edge uniformly at random in this example, however, the edges are stretched by at most a constant factor *in expectation,* justifying the hope that one may be able to randomly embed into a tree such that, for each pair of nodes, the *expected stretch* is small. A number of algorithms [6, 8, 9, 50] compute tree embeddings, culminating in the one by Fakcharoenphol, Rao, and Talwar (FRT) that achieves stretch $\mathrm{O}(\log n)$ in expectation [50], which is optimal [8]. Mendel and Schwob show how to sequentially sample from the FRT distribution in $\mathrm{O}(m \log^3 n)$ steps [110]. This upper bound has recently been improved to $\mathrm{O}(m \log n)$ w.h.p. by Blelloch et al. [20]. Both approaches match the trivial $\Omega(m)$ lower bound up to polylogarithmic factors. However, neither approach leads to a low-depth parallel algorithm.

Several parallel and distributed algorithms compute FRT trees [21, 65, 81]. These algorithms and ours have in common that they represent the embedding by LE lists, which were first introduced by Cohen [33]. In the parallel case, the state-of-the-art solution due to Blelloch et al. [21] achieves $\mathrm{O}(\log^2 n)$ depth and $\mathrm{O}(n^2 \log n)$ work, where the latter drops to $\mathrm{O}(n^2)$ if the ratio between the maximum and the minimum distance is exponentially bounded in $n$. However, Blelloch et al. assume the input to be given as an $n$-point *metric,* where the distance between two points can be queried at constant cost. Note that our approach is more general as a metric can be interpreted as a complete weighted graph of SPD 1; a single MBF-like iteration reproduces the result by Blelloch et al. Moreover, this point of view shows that the input required to achieve subquadratic work must be a sparse graph. For graph inputs, we are not aware of any algorithms achieving $\operatorname{polylog} n$ depth and a non-trivial work bound, i.e., not incurring the $\Omega(n^3)$ work caused by relying on matrix–matrix multiplication.

Furthermore, Blelloch et al. show how to obtain LE lists — through repeated use of an SSSP algorithm as black box — using $\mathrm{O}(D \log n)$ depth and $\mathrm{O}(W \log n)$ work, where $D$ and $W$ are the depth and work required by the SSSP algorithm [19]. As mentioned above, the LE lists readily yield an FRT tree. Since the SSSP routine is used for different source nodes, consistency with the triangle inequality needs to be established. We are, however, not aware of a, possibly approximate, SSSP algorithm which achieves that and still yields $D \in \operatorname{polylog} n$ and $W \in \tilde{\mathrm{O}}(m)$.

In the distributed setting, Khan et al. [81] demonstrate how to compute LE lists in $\mathrm{O}(\mathrm{SPD}(G) \log n)$ rounds in the CONGEST model. On the lower-bound side, trivially $\Omega(\mathrm{D}(G))$ rounds are required, where $\mathrm{D}(G)$ is the maximum hop distance — ignoring edge weights — between nodes. Even if $\mathrm{D}(G) \in \mathrm{O}(\log n)$, however, $\Omega(n^{1/2})$ rounds are necessary [38, 65]. Extending the algorithm by Khan et al., Ghaffari and Lenzen [65] show how to obtain a round complexity of $\tilde{\mathrm{O}}(\min\{n^{1/2+\varepsilon}, \mathrm{SPD}(G)\} + \mathrm{D}(G))$ for arbitrary $0 < \varepsilon$ at the expense of an expected stretch of $\mathrm{O}(\varepsilon^{-1} \log n)$. We partly build on these ideas; specifically, the construction of the graph $H$ in Chapter 15 can be seen as a generalization of the key technique from Ghaffari and Lenzen [65]. As detailed in Chapter 18, our framework of MBF-like algorithms subsumes the algorithms by Khan et al. and by Ghaffari and Lenzen. Leveraging further results [73, 103], we can improve upon them in terms of expected stretch and round complexity: We obtain a metric tree embedding with expected stretch $\mathrm{O}(\log n)$ that is computed in $\min\{\tilde{\mathrm{O}}((n^{1/2} + \mathrm{D}(G)) n^{\mathrm{o}(1)}), \tilde{\mathrm{O}}(\mathrm{SPD}(G))\}$ rounds in the CONGEST model.

## 12.3 Notation and Preliminaries

**Graphs** We consider weighted, undirected graphs $G = (V, E, \omega)$ with *nodes* $V$*, edges* $E$*,* and edge *weights* $\omega \colon E \to \mathbb{R}_{>0}$. Unless specified otherwise, we set $n := |V|$, $m := |E|$, and do not permit parallel edges. For an edge $e = \{v, w\} \in E$, we write $\omega(v, w) := \omega(e)$, $\omega(v, v) := 0$ for $v \in V$, and $\omega(v, w) := \infty$ for $\{v, w\} \notin E$. We assume that the ratio of the maximum and minimum edge weight is polynomially bounded in $n$, this is required by Cohen's hop-set construction [34]. Unless specified otherwise, we assume that $G$ is connected and given in the form of an adjacency list.

Let $p \subseteq E$ be a path, where paths are loop-free unless explicitly stated otherwise. $p$ has $|p|$ *hops,* and *weight*

$$\omega(p) := \sum_{e \in p} \omega(e). \tag{12.6}$$

For the nodes $v, w \in V$ let $\mathrm{P}(v, w, G)$ denote the set of paths from $v$ to $w$ in $G$ and $\mathrm{P}^h(v, w, G) := \{p \in \mathrm{P}(v, w, G) \mid |p| \leq h\}$ the set of such paths using at most $h$ hops, where we rule that $\emptyset \in \mathrm{P}^0(v, v, G)$ for all $v \in V$. We denote by

$$\mathrm{dist}^h(v, w, G) := \min\left\{\omega(p) \mid p \in \mathrm{P}^h(v, w, G)\right\} \tag{12.7}$$

the *h-hop distance* from $v$ to $w$, where $\min \emptyset := \infty$; the *distance* between $v$ and $w$ is $\mathrm{dist}(v, w, G) := \mathrm{dist}^n(v, w, G)$. The *shortest-path hop distance* between $v$ and $w$ is

$$\mathrm{hop}(v, w, G) := \min\{|p| \mid p \in \mathrm{P}(v, w, G) \land \omega(p) = \mathrm{dist}(v, w, G)\} \tag{12.8}$$

and

$$\mathrm{MHSP}(v, w, G) := \{p \in \mathrm{P}(v, w, G) \mid \omega(p) = \mathrm{dist}(v, w, G) \land |p| = \mathrm{hop}(v, w, G)\} \tag{12.9}$$

denotes all *min-hop shortest paths* from $v$ to $w$. Finally, the *Shortest-Path Diameter (SPD)* of $G$ is

$$\mathrm{SPD}(G) := \max_{v,w \in V} \mathrm{hop}(v, w, G) \tag{12.10}$$

and the unweighted *hop diameter* of $G$ is

$$\mathrm{D}(G) := \min\left\{h \in \mathbb{N} \mid \forall v, w \in V\colon\ \mathrm{dist}^h(v, w, G) < \infty\right\}. \tag{12.11}$$

**Hop Sets** A graph $G = (V, E, \omega)$ *contains a $(d, \hat{\varepsilon})$-hop set* if

$$\forall v, w \in V\colon \quad \mathrm{dist}^d(v, w, G) \leq (1 + \hat{\varepsilon})\,\mathrm{dist}(v, w, G), \tag{12.12}$$

i.e., if its $d$-hop distances $(1 + \hat{\varepsilon})$-approximate its distances. This definition is based on Cohen [34], who describes how to efficiently construct a set of weighted edges in order to establish this property for arbitrary weighted graphs.

**Metrics** A metric on a set $V$ captures the notion of distance between the elements of $V$. Examples are Euclidean distances in $\mathbb{R}^n$ and distances between nodes in a graph with positive edge weights. We complement the standard definition of metric and submetric with the term *$\alpha$-approximate metric.*

**Definition 12.2** (Metric, Submetric, and Approxmiate Metric)**.** *Given a set $V$, a function $d\colon V \times V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a* metric on $V$ *if, for all $u, v, w \in V$,* *(1) zero distance is equivalent to node identity*

$$d(v,w) = 0 \quad \Leftrightarrow \quad v = w, \tag{12.13}$$

*(2) distances are symmetric*

$$d(v,w) = d(w,v), \text{ and} \tag{12.14}$$

*(3) the* triangle inequality *holds*

$$d(u,w) \leq d(u,v) + d(v,w). \tag{12.15}$$

*Each $V' \subseteq V$ spans a* submetric *by restricting the domain of $d$ to $V' \times V'$. Consider $\alpha \in \mathbb{R}_{\geq 1}$ and a metric $d$ on $V$. We refer to $d'$ as* $\alpha$-approximate metric of $d$ *if*

*(1) $d'$ is a metric on $V$ and*

*(2) for all $v, w \in V$,*

$$d(v,w) \leq d'(v,w) \leq \alpha d(v,w). \tag{12.16}$$

Some authors explicitly require $d(v,w) \geq 0$ from a metric. This, however, follows from (12.13)–(12.15), as $2d(v,w) = d(v,w) + d(w,v) \geq d(v,v) = 0$ for all $v, w \in V$.

Note carefully that our definition of an approximate metric requires the approximation of the distance function to be a metric. For example, consider a graph $G = (V, E, \omega)$ that contains a $(d, \hat{\varepsilon})$-hop set. Clearly, $\operatorname{dist}(\cdot,\cdot,G)$ is a metric on $V$ and $\operatorname{dist}^d(\cdot,\cdot,G)$ $(1+\hat{\varepsilon})$-approximates $\operatorname{dist}(\cdot,\cdot,G)$ by definition. But as detailed in Observation 12.1, $\operatorname{dist}^d(\cdot,\cdot,G)$ is not necessarily an approximate *metric.*

**Algebraic Distance Computations** Refer to Chapter 2 for a brief specification of algebraic terms used below. Among others, we extensively use the min-plus semiring in the following chapters. It is a well-established tool to determine pairwise distances in a graph via the distance product, see, e.g., Alon et al. [5], Mohri [115], or Zwick [135].

**Definition 12.3** (Min-Plus Semiring)**.** *We refer to $\mathcal{S}_{\min,+} := (\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +)$ as the* min-plus semiring. *In literature, it is sometimes referred to as the* tropical semiring. *Let $G = (V, E, \omega)$ be a graph. The* adjacency matrix $A \in \mathcal{S}_{\min,+}^{V \times V}$ of $G$ *is given by*

$$(a_{vw}) := \begin{cases} 0 & \text{if } v = w, \\ \omega(v,w) & \text{if } \{v,w\} \in E, \text{ and} \\ \infty & \text{otherwise.} \end{cases} \tag{12.17}$$

The operations involved in matrix addition and multiplication are those of the underlying semiring $\mathcal{S} = (S, \oplus, \odot)$, i.e., for square matrices $A, B \in \mathcal{S}^{V \times V}$ we have

$$(A \oplus B)_{vw} = a_{vw} \oplus b_{vw} \text{ and} \tag{12.18}$$

$$(AB)_{vw} = \bigoplus_{u \in V} a_{vu} \odot b_{uw}. \tag{12.19}$$

For $A, B \in \mathcal{S}_{\min,+}^{V \times V}$ in particular, we obtain

$$(A \oplus B)_{vw} = \min \{a_{vw}, b_{vw}\} \text{ and} \tag{12.20}$$

$$(AB)_{vw} = \min_{u \in V} \{a_{vu} + b_{uw}\}, \tag{12.21}$$

Alon et al. refer to the latter as "funny matrix multiplication" [5]. Observe that the identity matrix over $\mathcal{S}_{\min,+}$, $I \in \mathcal{S}_{\min,+}^{V \times V}$, is

$$I_{vw} = \begin{cases} 0 & \text{if } v = w \text{ and} \\ \infty & \text{otherwise} \end{cases} \tag{12.22}$$

which is the adjacency matrix of a graph without edges. The abovementioned *distance product* $A^h$ is determined over $\mathcal{S}_{\min,+}$ and corresponds to $h$-hop distances, i.e., $(A^h)_{vw} = \operatorname{dist}^h(v, w, G)$. In particular, this corresponds to exact distances for $h \geq \operatorname{SPD}(G)$. This is a standard result [5, 35, 115, 135], however, we shed some light on it in Chapter 14 w.r.t. our notation and some additional techniques introduced in the following.

**Model of Computation** We are interested in NC and RNC algorithms. NC is the class of problems that can be solved by a Parallel Random-Access Machine (PRAM) with poly $n$ processors in polylog $n$ time; RNC additionally allows for randomization. Refer to Johnson [78] for a formal introduction.

Our perspective is that an algorithm defines a Directed Acyclic Graph (DAG), compare JáJá [77]. Input and constants constitute the sources of the DAG. Each non-source node of the DAG has constantly bounded fan-in and performs a basic instruction — such as addition, multiplication, comparison, etc. — on its inputs, where the inputs are either source nodes or recursively evaluated. Some predefined subset of the nodes marks the algorithm's output. We allow using independent randomness, hence the DAG is a random graph.

Given an instance of the problem, the *work* is the number of nodes of the DAG and the *depth* — parallel time — is its longest directed path [21, 22, 34, 77]. In the absence of read and write conflicts, the work is proportional to the time required by a single processor of uniform speed to complete the computation; the depth is the lowest time that can be achieved with an arbitrarily large number of processors. Note that work and depth may be random variables. Algorithms that achieve low — e.g., polylogarithmic — depth and a work close to good sequential-time algorithms are of interest because they are efficient on machines with a few as well as with many processors. We occasionally use the term *constant time* to refer to an operation with constantly bounded work and depth.

The problems computable in polylogarithmic parallel time on a polynomial number of processors in the Concurrent Read Concurrent Write (CRCW), Concurrent Read Exclusive Write (CREW), and Exclusive Read Exclusive Write (EREW) PRAM models constitute subsets of NC, which in turn is a subset of P [78]. However, the measure of depth is not robust across these models [22], which makes the above work/depth perspective on the NC and RNC classes convenient. Furthermore, the work/depth perspective avoids distraction from memory locality, synchronization primitives, communication between processors, scheduling, and many other aspects of various models for parallel computations [77].

When making probabilistic statements, we require that they hold for all instances, i.e., that the respective probability bounds are satisfied after fixing an arbitrary instance. Further, we assume that a single register can store any number computed throughout the algorithm with sufficient precision. As we are interested in approximation algorithms and assume polynomially bounded edge weights, $\mathrm{O}(\log n)$ bits yield sufficient precision.

# CHAPTER 13

# MBF-like Algorithms

The Moore-Bellman-Ford (MBF) algorithm [16, 54, 116] is both fundamental and elegant. In its classical form, see Algorithm 13.1, it solves the SSSP problem: Given a weighted graph $G = (V, E, \omega)$ and a source node $s \in V$, determine, for all $v \in V$, $\operatorname{dist}(s, v, G)$ and the predecessor on a shortest path from $s$ to $v$. The MBF algorithm comprises $n - 1$ iterations.[1] It maintains the invariant that by the end of the $i$-th iteration, each node $v \in V$ knows $\operatorname{dist}^i(s, v, G)$, the $i$-hop distance from $s$ to $v$, as well as the predecessor on an according $i$-hop shortest path. The initialization is $\operatorname{dist}^0(s, v, G)$ which is 0 for $v = s$ and $\infty$ everywhere else. Given $\operatorname{dist}^i(\cdot, \cdot, G)$, $\operatorname{dist}^{i+1}(\cdot, \cdot, G)$ is determined by *relaxing* each edge: If $\operatorname{dist}^i(s, w, G) + \omega(w, v)$ is smaller than the shortest $s$-$v$ path seen so far, update accordingly. All $(i + 1)$-hop shortest $s$-$w$-paths are either $i$-hop shortest $s$-$w$-paths or composed of an $i$-hop shortest $s$-$v$-path followed by the edge $\{v, w\}$. As the MBF algorithm explores all edges in all iterations, $i$ iterations yield $i$-hop $s$-$v$ distances. Since every edge is relaxed in every iteration, a sequential implementation of the MBF algorithm takes $\mathrm{O}(nm)$ time [35].

Dijkstra's algorithm, see Algorithm 13.2, is the classical alternative to the MBF algorithm for solving SSSP [41]. It builds a shortest-path tree from $s$, adding the closest unprocessed node in each iteration. To this end, it maintains a queue $Q$ of unprocessed nodes. For the processed nodes $V \setminus Q$, a shortest-path tree and $\operatorname{dist}(s, v, G)$ is known; all unprocessed nodes have at least the same distance to $s$ as the processed ones. Initially, we have $Q = V$, $\operatorname{dist}(s, s, G) = 0$, and $\operatorname{dist}(s, v, G) = \infty$ for all $v \neq s$. In each iteration, Dijkstra's algorithm processes the node in $Q$ that is closest to $s$. No shortest path can return from $v \in Q$ to $V \setminus Q$, hence $\operatorname{dist}(s, v, G)$ can be fixed when $v$ leaves the queue and its incident edges are relaxed. In the sequential setting, this can be implemented using $\mathrm{O}(n \log n + m)$ time [35].

Observe that the invariants of the MBF algorithm and of Dijkstra's algorithm are quite different. Dijkstra's algorithm "flood fills" the graph and guarantees maintaining a shortest-path tree by processing nodes in the correct order, using a global priority queue. The MBF algorithm, on the other hand, relaxes all edges in all iterations which is possible without global information: An edge $\{v, w\}$ can be relaxed knowing only $\operatorname{dist}(s, v, G)$, $\operatorname{dist}(s, w, G)$, and $\omega(v, w)$.

This has consequences for parallelizing these algorithms. While the inner loop of both algorithms — which relaxes either all or a subset of the edges — easily parallelizes, the outer loop is more challenging. For both algorithms, in order to maintain hope for sublinear depth, either the number of iterations in the outer loop must be reduced or some of these iterations have to be done in parallel. Regarding the MBF algorithm, this means that we need the $h$-hop distances to be good approximations for the $n$-hop distances; this problem is addressed by hop sets [34, 48, 73], making the MBF algorithm

[1] $\operatorname{SPD}(G)$ iterations suffice, but $\operatorname{SPD}(G) = n - 1$ is possible.

```
input: Weighted graph $G = (V, E, \omega)$, source node $s \in V$
output: Shortest paths to $s$ (encoded in distance($\cdot$) and predecessor($\cdot$))
 1: INIT($s, V$)
 2: for $i \leftarrow 1, \dots, n-1$ do                    ▷ $n - 1 \geq \mathrm{SPD}(G)$ iterations
 3:     for each $\{v, w\} \in E$ do
 4:         RELAX($\omega, v, w$)
 5:         RELAX($\omega, w, v$)
 6:     end for
 7: end for

 8: function RELAX($\omega, v, w$)
 9:     if distance($w$) > distance($v$) + $\omega(v, w)$ then
10:         distance($w$) $\leftarrow$ distance($v$) + $\omega(v, w)$
11:         predecessor($w$) $\leftarrow v$
12:     end if
13: end function

14: function INIT($s, V$)                              ▷ Initialize with $\mathrm{dist}^0(s, \cdot, G)$
15:     for each $v \in V$ do
16:         distance($v$) $\leftarrow \infty$
17:         predecessor($v$) $\leftarrow \bot$
18:     end for
19:     distance($s$) $\leftarrow 0$
20: end function
```

**Algorithm 13.1:** The Moore-Bellman-Ford (MBF) algorithm [16, 54, 116], adapted from Cormen et al. [35].

```
input: Weighted graph $G = (V, E, \omega)$, source node $s \in V$
output: Shortest paths to $s$ (encoded in distance($\cdot$) and predecessor($\cdot$))
 1: INIT($s, V$)
 2: $Q \leftarrow V$
 3: while $Q \neq \emptyset$ do                                   ▷ $n$ iterations
 4:     $v \leftarrow \arg\min_{v \in Q}$ distance($v$)
 5:     $Q \leftarrow Q \setminus \{v\}$
 6:     for each $\{v, w\} \in E$ do
 7:         RELAX($\omega, v, w$)
 8:     end for
 9: end while
```

**Algorithm 13.2:** Dijkstra's algorithm [41], adapted from Cormen et al. [35].

a benevolent candidate for parallelization.

The queue in Dijkstra's algorithm, however, is an inherently sequential feature: Fewer than $n$ iterations imply that some subset of the nodes has not been visited, making it hard to guarantee that $\operatorname{dist}(s, \cdot, G)$ is approximated sufficiently well. Hence, multiple queued vertices have to be processed in parallel which is, e.g., addressed by the $\Delta$-stepping algorithm [113] of which there are efficient parallel implementations [105]. It performs well on certain classes of random graphs and on graphs of low degree, like road networks, but its worst-case guarantee is no better than $\tilde{\mathrm{O}}(dL)$ time in the PRAM model, where $d$ is the maximum vertex degree and $L$ the maximum weight of a shortest $s$-$v$-path. Hence good performance as well as $dL \in \Theta(nm)$ are possible, depending on the setting.

Over the years, numerous algorithms emerged that use schemes similar to the MBF algorithm for distributing information [65, 73, 81, 101, 102, 103, 104]. It is natural to ask for a characterization that captures all these algorithms. In this chapter, we propose such a characterization: the class of *MBF-like algorithms.* With parallel and distributed algorithms in mind, we take a rather distributed perspective, interpreting nodes as actors that share information across edges.

Observe that we introduce a level of complexity that, on first sight, may not seem justified in the light of the simplicity of Algorithm 13.1. It proves, however, capable of capturing many variants of the MBF algorithm; we demonstrate this in Chapter 14 and give an example below. Furthermore, our abstraction allows us to maintain a clean separation between our techniques — primarily, the oracle for MBF-like algorithms in Chapter 15 — and our results. While most proposed techniques originally aimed at the development of parallel FRT algorithms (Chapter 17), they generalize to distributed FRT embeddings (Chapter 18) and parallel metric and submetric approximations (Chapter 16).

Our abstraction is the following:

(1) An initial state vector $x^{(0)} \in M^V$ contains the information initially known to each node. $M$ is the set of possible node states and can be seen as a "data type" capable of holding the information that the MBF-like algorithm can produce at an arbitrary node.

(2) In *iteration* $i + 1$ each node first *propagates* the information available to it from the last iteration, i.e., $x_v^{(i)} \in M$, along all incident edges as well as to itself.

(3) All nodes then *aggregate* the received information. This and the previous step correspond to relaxing all edges incident to a node $v$ in Algorithm 13.1.

(4) Finally, irrelevant information is *filtered* out before moving on to the next iteration. This is not required in Algorithm 13.1 as it never generates irrelevant information, but crucial to many of its more involved generalizations like the following example.

As a concrete example consider $k$-SSP, the task of determining for each node the list of its $k$ closest nodes. To this end, one needs to consider all nodes as sources and at the same time be able to locally discard distances that are too large, otherwise one solves APSP. Nodes store values in $(\mathbb{R}_{\geq 0} \cup \{\infty\})^V$, so that in iteration $i$ each node $v \in V$ can store $\operatorname{dist}^i(v, w, G) \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ for all $w \in V$.

Initially, $x_{vw}^{(0)}$ is 0 if $v = w$ and $\infty$ everywhere else (the 0-hop distances). *Propagating* these distances over an edge of weight $\omega(e)$ in iteration $i + 1$ means uniformly increasing them by $\omega(e)$. This corresponds to appending $e$ to the $i$-hop shortest paths determined in the previous iteration; the state $x_v^{(i)}$ at node $v$ from the previous iteration is memorized by $v$ propagating $x_v^{(i)}$ over an "edge" of weight 0 to itself. During *aggregation,* each node picks, for each target node, the smallest distance reported so far.

This is costly, since each node might learn non-$\infty$ distances values for *all* other nodes. To increase efficiency, we *filter* out, in each iteration and at each node, all source–distance pairs but the $k$ pairs with the smallest distance. Note that filtering makes the difference between solving APSP — and only in the end discarding all but the $k$ closest sources per node — and the variant that continuously discards irrelevant distances. Both versions are correct, but the latter (with $\tilde{\Theta}(mk)$ work per iteration) is more efficient than the former (with $\tilde{\Theta}(mn)$ work per iteration). We formalize $k$-SSP in Example 14.5.

The crucial characteristics exploited by this idea are the following.

(1) Propagation and aggregation are interchangeable in the sense that it makes no difference whether two pieces of information are propagated separately or as a single aggregated piece of information.

(2) As motivated above, filtering or not filtering after aggregation has no impact on the correctness of an algorithm. Omitting the filtering step, however, may increase work or the amount of information propagated along edges; the latter is relevant in distributed algorithms in the CONGEST model.

In this chapter, we formalize this approach for later use in more advanced algorithms. To this end, we develop a characterization of MBF-like algorithms in Sections 13.1–13.3 and establish basic properties in Section 13.4. We verify that our characterization captures a wide variety of known algorithms in Chapter 14. Refer to Chapter 2 for basic algebraic definitions.

## 13.1 Propagation and Aggregation

For the initial discussion, consider the min-plus semiring $\mathcal{S}_{\min,+}$ and the according adjacency matrix, both from Definition 12.3; we generalize below. Let $M$ be the set of node states, i.e., the possible values that an MBF-like algorithm can store at a single vertex. We represent propagation of $x \in M$ over an edge of weight $s \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ by $s \odot x$, where $\odot\colon \mathbb{R}_{\geq 0} \cup \{\infty\} \times M \to M$, and aggregation of $x, y \in M$ by $x \oplus y$, where $\oplus\colon M \times M \to M$. Filtering is deferred to Section 13.2. Concerning the aggregation of information, we demand that $\oplus$ is associative and has a neutral element $\perp \in M$ encoding "no available information," hence $(M, \oplus)$ is a semigroup with neutral element $\perp$ (see Definition 2.1). Furthermore, we require for all $s, t \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ and $x, y \in M$ (note that we "overload" $\oplus$ and $\odot$) that

$$0 \odot x = x, \tag{13.1}$$

$$\infty \odot x = \perp, \tag{13.2}$$

$$s \odot (x \oplus y) = (s \odot x) \oplus (s \odot y), \tag{13.3}$$

$$(s \oplus t) \odot x = (s \odot x) \oplus (t \odot x), \text{ and} \tag{13.4}$$

$$(s \odot t) \odot x = s \odot (t \odot x). \tag{13.5}$$

Our requirements are quite natural: Equations (13.1) and (13.2) state that propagating information over zero distance (e.g. keeping it at a vertex) does not alter it and that propagating it infinitely far away (i.e., "propagating" it over a non-existing edge) means losing it, respectively. Note that 0 and $\infty$ are the neutral elements w.r.t. $\odot$ and $\oplus$ in $\mathcal{S}_{\min,+}$. Equation (13.3) says that propagating aggregated information is equivalent to aggregating propagated information along identical distances, Equation (13.4) means that propagating information over only the shorter of two edges is equivalent to moving it along both edges and then aggregating it (information "becomes obsolete" with increasing distance), and Equation (13.5) states that propagation steps may be merged.

Altogether, this is equivalent to demanding that $\mathcal{M} = (M, \oplus, \odot)$ is a zero-preserving semimodule (see Definition 2.3) over $\mathcal{S}_{\min,+}$. A straightforward choice of $\mathcal{M}$ is the direct product of $|V|$ copies of $\mathbb{R}_{\geq 0} \cup \{\infty\}$, which is suitable for many of the applications we consider.

**Definition 13.1** (Distance Map)**.** *Let $\mathcal{D} := ((\mathbb{R}_{\geq 0} \cup \{\infty\})^V, \oplus, \odot)$, where $\oplus$ and $\odot$ are defined such that for all $s \in \mathcal{S}_{\min,+}$ and $x, y \in \mathcal{D}$,*

$$(x \oplus y)_v := x_v \oplus y_v = \min\{x_v, y_v\} \text{ and} \tag{13.6}$$

$$(s \odot x)_v := s \odot x_v = s + x_v, \tag{13.7}$$

*be the* distance map semimodule.

**Corollary 13.2.** *By Lemma 2.4, $\mathcal{D}$ is a zero-preserving semimodule over $\mathcal{S}_{\min,+}$ with zero $\perp = (\infty, \dots, \infty)^\top$.*

Distance maps can be represented by only storing the non-$\infty$ distances (and their indices from $V$). This is of interest when there are few non-$\infty$ entries, which can, e.g., be ensured by filtering (see below). In the following, we denote by $|x|$ the number of non-$\infty$ entries of $x \in \mathcal{D}$. The following lemma shows that this representation allows efficient aggregation.

**Lemma 13.3.** *Suppose $x_1, \dots, x_n \in \mathcal{D}$ are stored in lists of index–distance pairs as above. Then $\bigoplus_{i=1}^n x_i$ can be computed with $\mathrm{O}(\log n)$ depth and $\mathrm{O}(\sum_{i=1}^n |x_i| \log n)$ work.*

*Proof.* We sort $\bigcup_{i=1}^n \{(v, x_{iv}) \mid x_{iv} \neq \infty\}$ (with duplicates) in ascending lexicographical order. This can be done in parallel with $\mathrm{O}(\log(\sum_{i=1}^n |x_i|)) \subseteq \mathrm{O}(\log n)$ depth and $\mathrm{O}(\sum_{i=1}^n |x_i| \log n)$ work [3]. Then we delete each pair for which the next smaller pair has the same index; the resulting list hence contains, for each $v \in V$ for which there is a non-$\infty$ value in some list $x_i$, the minimum such value. As the last operation is easy to implement with $\mathrm{O}(\log n)$ depth and $\mathrm{O}(\sum_{i=1}^n |x_i| \log n)$ work, the claim follows. $\square$

While $\mathcal{S}_{\min,+}$ and $\mathcal{D}$ suffice for many applications and are suitable to convey our ideas, it is sometimes necessary to use different semirings and semimodules to unfold the full potential of our framework. We elaborate on this in Chapter 14. Hence, rather than confining the discussion to semimodules over $\mathcal{S}_{\min,+}$, in the following we make general statements about an arbitrary semimodule $\mathcal{M} = (M, \oplus, \odot)$ over an arbitrary semiring $\mathcal{S} = (S, \oplus, \odot)$ wherever it does not obstruct the presentation. It is, however, helpful to keep $\mathcal{S} = \mathcal{S}_{\min,+}$ and $\mathcal{M} = \mathcal{D}$ in mind.

## 13.2 Filtering

MBF-like algorithms achieve efficiency by maintaining and propagating — instead of the full amount of information nodes are exposed to — only a filtered (small) representative of the information they obtained. Our goal in this section is to capture the properties a filter must satisfy. We start with a congruence relation, i.e., an equivalence relation compatible with propagation and aggregation, on $\mathcal{M}$. A filter $r\colon \mathcal{M} \to \mathcal{M}$ is a projection mapping all members of an equivalence class to the same representative within that class, compare Definition 13.6.

**Definition 13.4** (Congruence Relation)**.** *Let $\mathcal{M} = (M, \oplus, \odot)$ be a semimodule over the semiring $\mathcal{S}$ and $\sim$ an equivalence relation on $M$. We call $\sim$ a* congruence relation on $\mathcal{M}$ *if and only if*

$$\forall s \in \mathcal{S}, \forall x, x' \in \mathcal{M}\colon \quad x \sim x' \Rightarrow sx \sim sx' \tag{13.8}$$
$$\forall x, x', y, y' \in \mathcal{M}\colon \quad x \sim x' \wedge y \sim y' \Rightarrow x \oplus y \sim x' \oplus y'. \tag{13.9}$$

A congruence relation induces a quotient semimodule.

**Observation 13.5.** *Denote by $[x]$ the equivalence class of $x \in \mathcal{M}$ under the congruence relation $\sim$ on the semimodule $\mathcal{M}$. Set $M/_\sim := \{[x] \mid x \in \mathcal{M}\}$. Then $\mathcal{M}/_\sim := (M/_\sim, \oplus, \odot)$ is a semimodule with the operations $[x] \oplus [y] := [x \oplus y]$ and $s \odot [x] := [sx]$.*

An MBF-like algorithm performs efficient computations by implicitly operating on this quotient semimodule, i.e., on suitable — small — representatives of the equivalence classes. Such representatives are obtained in the filtering step using a *representative projection,* also referred to as *filter.* We refer to this step as filtering since, in all our applications and examples, it discards a subset of the available information that is irrelevant to the problem at hand.

**Definition 13.6** (Representative Projection)**.** *Let $\mathcal{M}$ be a semimodule over the semiring $\mathcal{S}$ and $\sim$ a congruence relation on $\mathcal{M}$. Then $r\colon \mathcal{M} \to \mathcal{M}$ is a* representative projection w.r.t. $\sim$ *if and only if*

$$\forall x \in \mathcal{M}\colon \quad x \sim r(x) \tag{13.10}$$
$$\forall x, y \in \mathcal{M}\colon \quad x \sim y \Rightarrow r(x) = r(y). \tag{13.11}$$

**Observation 13.7.** *A representative projection is a projection, i.e., $r^2 = r$.*

In the following, we typically first define a suitable projection $r$; this projection in turn defines equivalence classes $[x] := \{y \in \mathcal{M} \mid r(x) = r(y)\}$. The following lemma is useful when we need to show that equivalence classes defined this way yield a congruence relation, i.e., are suitable for MBF-like algorithms.

**Lemma 13.8.** *Let $\mathcal{M}$ be a semimodule over the semiring $\mathcal{S}$, let $r\colon \mathcal{M} \to \mathcal{M}$ be a projection, and for $x, y \in \mathcal{M}$, let $x \sim y :\Leftrightarrow r(x) = r(y)$. Then $\sim$ is a congruence relation with representative projection $r$ if:*

$$\forall s \in \mathcal{S}, \forall x, x' \in \mathcal{M}\colon \quad r(x) = r(x') \Rightarrow r(sx) = r(sx'), \text{ and} \tag{13.12}$$
$$\forall x, x', y, y' \in \mathcal{M}\colon \quad r(x) = r(x') \wedge r(y) = r(y') \Rightarrow r(x \oplus y) = r(x' \oplus y'). \tag{13.13}$$

*Proof.* Obviously, $\sim$ is an equivalence relation and $r$ fulfills (13.10) and (13.11). Conditions (13.8) and (13.9) directly follow from the preconditions of the lemma. $\square$

An MBF-like algorithm has to behave in a compatible way for all vertices in that each vertex follows the same propagation, aggregation, and filtering rules. This induces a semimodule structure on the (possible) state vectors of the algorithm in a natural way.

**Definition 13.9** (Power Semimodule)**.** *Given a node set $V$ and a zero-preserving semimodule $\mathcal{M} = (M, \oplus, \odot)$ over the semiring $\mathcal{S}$, we define $\mathcal{M}^V := (M^V, \oplus, \odot)$ by applying the operations of $\mathcal{M}$ coordinatewise, i.e., $\forall x, y \in M^V$, $\forall s \in \mathcal{S}$, and $\forall v \in V$:*

$$(x \oplus y)_v := x_v \oplus y_v \text{ and} \tag{13.14}$$

$$(s \odot x)_v := s \odot x_v. \tag{13.15}$$

*Furthermore, we denote the componentwise application of a representative projection $r$ by $r^V$, such that for all $x \in M^V$ and $v \in V$:*

$$r^V(x)_v := r(x_v), \tag{13.16}$$

*inducing the equivalence relation $\sim$ on $\mathcal{M}^V$ via*

$$x \sim y \quad :\Leftrightarrow \quad \forall v \in V \colon x_v \sim y_v. \tag{13.17}$$

**Observation 13.10.** *By Lemma 2.4, $\mathcal{M}^V$ is a zero-preserving semimodule over $\mathcal{S}$ with zero $\perp^V := (\perp, \dots, \perp)^\top \in \mathcal{M}^V$, where $\perp$ is the zero of $\mathcal{M}$. The equivalence relation $\sim$ induced by $r^V$ is a congruence relation on $\mathcal{M}^V$ with representative projection $r^V$.*

## 13.3 The Class of MBF-like Algorithms

The following definition connects the properties introduced and motivated above.

**Definition 13.11** (MBF-like Algorithm)**.** *A* Moore-Bellman-Ford-like (MBF-like) algorithm *$\mathcal{A}$ is determined by*

*(1) a zero-preserving semimodule $\mathcal{M}$ over a semiring $\mathcal{S}$,*

*(2) a congruence relation on $\mathcal{M}$ with representative projection $r \colon \mathcal{M} \to \mathcal{M}$, and*

*(3) initial values $x^{(0)} \in \mathcal{M}^V$ for the nodes (which may depend on the input graph).*

*In a graph $G$ with adjacency matrix $A$, $h$ iterations of $\mathcal{A}$ determine*

$$\mathcal{A}^h(G) := x^{(h)} := r^V A^h x^{(0)}. \tag{13.18}$$

*If $\mathcal{A}$ reaches a fixed point after $h \in \mathbb{N}_0$ iterations in $G$, i.e., if $x^{(h)} = x^{(h+1)}$, we abbreviate $\mathcal{A}(G) := A^h(G)$.*

Note that the definition of the adjacency matrix $A \in \mathcal{S}^{V \times V}$ depends on the choice of the semiring $\mathcal{S}$. For the standard choice of $\mathcal{S} = \mathcal{S}_{\min,+}$, which suffices for all our core results, we define $A$ in Equation (12.17); examples using $\mathcal{S}_{\min,+}$ as well as other semirings and the associated adjacency matrices are discussed in Chapter 14.

The $(i+1)$-th iteration of an MBF-like algorithm $\mathcal{A}$ determines $x^{(i+1)} := r^V A x^{(i)}$ (propagate, aggregate, and filter). Thus, $h$ iterations yield $(r^V A)^h x^{(0)}$ which we show to be identical to $r^V A^h x^{(0)}$ in Corollary 13.20 of Section 13.4.

**Observation 13.12** (Fixed Points)**.** *Over $\mathcal{S}_{\min,+}$ and for the adjacency matrix $A$ from Equation* (12.17)*, $\mathcal{A}$ reaches a fixed point after at most* SPD($G$) *iterations. This is due to the fact that $A^h_{vw} = \operatorname{dist}^h(v,w,G)$, compare Lemma 14.1. In fact, most examples that we present reach a fixed point after at most $n-1$ iterations because they explore loop-free paths. Not all MBF-like algorithms, however, have a fixed point. As an example, consider distances w.r.t. paths that have* exactly *$h$ hops in a cycle. These distances can be determined by an MBF-like algorithm when replacing the entries on the diagonal of $A$ in Equation* (12.17) *by $\infty$, forcing nodes to discard their state from the previous iteration.*

## 13.4 Equivalence of Iterations

In the following chapters, we need statements like "one iteration of an MBF-like algorithm in the graph $H$ is equivalent to $d$ appropriately arranged iterations in the graph $G$." The first step of formalizing that statement is Definition 13.9 which lifts the notion of state equivalence from single nodes (on $\mathcal{M}$) to the entirety of a graph's nodes (on $\mathcal{M}^V$). We complete the formalization using the following steps.

(1) We introduce *Simple Linear Functions (SLFs),* the functions needed to iterate MBF-like algorithms. Those are exactly the matrices $\mathcal{S}^{V\times V}$, adjacency and other. The set of SLFs together with addition and concatenation of functions is isomorphic to the matrix semiring over $\mathcal{S}$ and forms a proper subset of the linear functions on $\mathcal{M}^V$ (Definition 13.13, Remark 13.14, and Lemma 13.15).

(2) The next step is to observe that SLFs are well-behaved w.r.t. the equivalence classes $\mathcal{M}^V/_\sim$ of node states: SLFs map equivalent node states to equivalent node states (Lemma 13.16).

(3) We introduce the notion of equivalent functions and, by extension, equivalent iterations. Two functions are equivalent if and only if they map equivalent inputs to equivalent outputs. Equivalence classes of SLFs yield the functions required for the study of MBF-like algorithms. These form a semiring of (a subset of) the functions on $\mathcal{M}^V/_\sim$ (Observation 13.17 and Theorem 13.18).

(4) Next, we observe that $r^V \sim \operatorname{id}$, formalizing the concepts of "operating on equivalence classes of node states" and "filtering being optional w.r.t. correctness" (Corollary 13.20).

(5) Finally, we remark that our restriction to SLFs and componentwise filtering is required by the framework at hand (Remarks 13.21 and 13.22).

An SLF $f$ is "simple" in the sense that it corresponds to matrix–vector multiplications, i.e., maps $x \in \mathcal{M}^V$ such that $(f(x))_v$ is a linear combination of the coordinates $x_w$, $w \in V$, of $x$.

**Definition 13.13** (Simple Linear Function)**.** *Let $\mathcal{M}$ be a semimodule over the semiring $\mathcal{S}$. Each matrix $A \in \mathcal{S}^{V\times V}$ defines a* Simple Linear Function (SLF) *$A\colon \mathcal{M}^V \to \mathcal{M}^V$ (and vice versa) by*

$$A(x)_v := (Ax)_v = \bigoplus_{w\in V} a_{vw}x_w. \tag{13.19}$$

Thus, each iteration of an MBF-like algorithm is an application of an SLF given by an (adjacency) matrix followed by an application of the filter $r^V$. Observe that asymmetric matrices (for directed graphs) or matrices with non-zero entries on the diagonal are not a problem for our framework. Cohen's hop-set construction, however, explicitly requires an undirected graph [34]. Hence, we inherit the same requirement wherever we rely on Cohen's hop sets, most importantly regarding the parallel computations of approximate metrics and FRT embeddings in Chapters 16 and 17.

**Remark 13.14** (Non-Simple Linear Function). *Not all linear*[2] *functions on $\mathcal{M}^V$ are SLFs. Choose $V = \{1, 2\}$, $\mathcal{S} = \mathcal{S}_{\min,+}$, and $\mathcal{M} = \mathcal{D}$. Consider $f\colon \mathcal{M}^V \to \mathcal{M}^V$ given by*

$$f\begin{pmatrix}(x_{11}, x_{12})\\(x_{21}, x_{22})\end{pmatrix} := \begin{pmatrix}(x_{11} \oplus x_{12}, \infty)\\ \perp\end{pmatrix}. \tag{13.20}$$

*While $f$ is linear, $f(x)_1$ is not a linear combination of $(x_{11}, x_{12})$ and $(x_{21}, x_{22})$. Hence, $f$ is not an SLF. We demonstrate in Remark 13.21, however, that our techniques require the restriction to SLFs.*

In the following, fix a semiring $\mathcal{S}$, a semimodule $\mathcal{M}$ over $\mathcal{S}$, a congruence relation $\sim$ on $\mathcal{M}$, and a representative projection $r$ w.r.t. $\sim$. Let $A, B \in \mathcal{S}^{V\times V}$ be SLFs. We write $A(x)$ for function application, $Ax$ for matrix–vector multiplication, and $(A \oplus B)(x) := A(x) \oplus B(x)$ and $(A \circ B)(x) := A(B(x))$ for the addition and concatenation of SLFs, respectively. In Lemma 13.15, we show that matrix addition and multiplication are equivalent to the addition and concatenation of SLFs. It follows that the SLFs form a semiring that is isomorphic to the matrix semiring over $\mathcal{S}$. Hence, we may use $A(x)$ and $Ax$ interchangeably in the following.

**Lemma 13.15.** *$\mathcal{F} := (\mathcal{S}^{V\times V}, \oplus, \circ)$, where $\oplus$ denotes the addition of functions and $\circ$ their concatenation, is a semiring. Furthermore, $\mathcal{F}$ is isomorphic to the matrix semiring $(\mathcal{S}^{V\times V}, \oplus, \odot)$ over $\mathcal{S}$, i.e., for all $A, B \in \mathcal{S}^{V\times V}$ and $x \in \mathcal{M}^V$,*

$$(A \oplus B)(x) = (A \oplus B)x \text{ and} \tag{13.21}$$

$$(A \circ B)(x) = ABx. \tag{13.22}$$

*Proof.* Let $A, B \in \mathcal{S}^{V\times V}$ and $x \in \mathcal{M}^V$ be arbitrary. Regarding (13.21) and (13.22), observe that we have

$$(A \oplus B)x = Ax \oplus Bx = A(x) \oplus B(x) = (A \oplus B)(x) \text{ and} \tag{13.23}$$

$$ABx = A(Bx) = A(B(x)) = (A \circ B)(x), \tag{13.24}$$

respectively, i.e., addition and concatenation of SLFs are equivalent to addition and multiplication of their respective matrices. It follows that $\mathcal{F}$ is isomorphic to the matrix semiring $(\mathcal{S}^{V\times V}, \oplus, \odot)$ and hence $\mathcal{F}$ is a semiring as claimed. ☐

Recall that MBF-like algorithms project node states to appropriate equivalent node states and that SLFs correspond to MBF-like iterations. Hence, it is important that

[2]A linear function $f\colon \mathcal{M} \to \mathcal{M}$ on the semimodule $\mathcal{M}$ over the semiring $\mathcal{S}$ satisfies, for all $x, y \in \mathcal{M}$ and $s \in \mathcal{S}$, that $f(x \oplus y) = f(x) \oplus f(y)$ and $f(s \odot x) = s \odot f(x)$.

SLFs are well-behaved w.r.t. the equivalence classes $\mathcal{M}^V/_\sim$ of node states. Formally, we call a function $A\colon \mathcal{M}^V \to \mathcal{M}^V$ — SLF or other — *well-behaved w.r.t.* $\sim$ if and only if

$$\forall x, x' \in \mathcal{M}^V\colon \quad x \sim x' \quad \Rightarrow \quad A(x) \sim A(x'). \tag{13.25}$$

**Lemma 13.16.** *Let $A \in \mathcal{S}^{V \times V}$ be an SLF. Then $A$ is well-behaved w.r.t. $\sim$.*

*Proof.* First, for $k \in \mathbb{N}$, let $x_1, \dots, x_k, x'_1, \dots, x'_k \in \mathcal{M}$ be such that $x_i \sim x'_i$ for all $1 \leq i \leq k$. We show that for all $s_1, \dots, s_k \in \mathcal{S}$ it holds that

$$\bigoplus_{i=1}^{k} s_i x_i \sim \bigoplus_{i=1}^{k} s_i x'_i. \tag{13.26}$$

We argue that (13.26) holds by induction over $k$. For $k = 1$, the claim trivially follows from Equation (13.8). Regarding $k \geq 2$, suppose the claim holds for $k-1$. Since $x_k \sim x'_k$, we have that $s_k x_k \sim s_k x'_k$ by Equation (13.8). The induction hypothesis yields $\bigoplus_{i=1}^{k-1} s_i x_i \sim \bigoplus_{i=1}^{k-1} s_i x'_i$. Hence,

$$\bigoplus_{i=1}^{k} s_i x_k = \left( \bigoplus_{i=1}^{k-1} s_i x_i \right) \oplus s_k x_k \overset{(13.9)}{\sim} \left( \bigoplus_{i=1}^{k-1} s_i x'_i \right) \oplus s_k x'_k = \bigoplus_{i=1}^{k} s_i x'_k. \tag{13.27}$$

As for the original claim, let $v \in V$ be arbitrary and note that we have

$$(Ax)_v = \bigoplus_{w \in V} a_{vw} x_v \overset{(13.26)}{\sim} \bigoplus_{w \in V} a_{vw} x'_v = (Ax')_v. \tag{13.28}$$

□

Hence, each SLF $A \in \mathcal{S}^{V \times V}$ not only defines a function $A\colon \mathcal{M}^V \to \mathcal{M}^V$, but also a function $A\colon \mathcal{M}^V/_\sim \to \mathcal{M}^V/_\sim$ with $A[x] := [Ax]$. This is useful, since MBF-like algorithms implicitly operate on $\mathcal{M}^V/_\sim$. As a last preliminary for Theorem 13.18, we need the concept of equivalent functions. To this end, we rule for all functions $A, B\colon \mathcal{M}^V \to \mathcal{M}^V$ that are well-behaved w.r.t. $\sim$ by Equation (13.25) — SLFs as well as other functions — that

$$A \sim B \quad :\Leftrightarrow \quad \forall x \in \mathcal{M}^V : A(x) \sim B(x), \tag{13.29}$$

i.e., that they are equivalent if and only if they yield equivalent results when presented with the same input. Let $[A]$ be the set of functions equivalent to $A$. This directly leads us to the observation that $[A]([x]) := [A(x)]$ is well-defined.

**Observation 13.17.** *Let $A \sim A'\colon \mathcal{M}^V \to \mathcal{M}^V$ be functions that are well-behaved w.r.t. $\sim$. Then, for all $x \sim x' \in \mathcal{M}^V$:*

$$A(x) \overset{(13.25)}{\sim} A(x') \overset{(13.29)}{\sim} A'(x'), \tag{13.30}$$

*i.e., $[A]([x]) := [A(x)]$ is well-defined.*

Regarding SLFs, we may partition $\mathcal{F}$ into equivalence classes $\mathcal{S}^{V \times V}/_\sim$. In Theorem 13.18, we show that these equivalence classes of functions, together with summation and concatenation, yield a semiring $\mathcal{F}/_\sim$. As MBF-like algorithms implicitly work on $\mathcal{M}^V/_\sim$, we obtain with $\mathcal{F}/_\sim$ precisely the structure that may be used to manipulate the state of MBF-like algorithms, which we leverage in Chapter 15.

**Theorem 13.18.** *$\mathcal{F}/_\sim := (\mathcal{S}^{V\times V}/_\sim, \oplus, \circ)$, where $\oplus$ denotes the addition and $\circ$ the concatenation of functions, is a semiring of functions on $\mathcal{M}^V/_\sim$ with*

$$[A] \oplus [B] = [A \oplus B] \text{ and} \tag{13.31}$$
$$[A] \circ [B] = [AB]. \tag{13.32}$$

*Proof.* Consider arbitrary $A, B \in \mathcal{S}^{V\times V}$ and $x \in \mathcal{M}^V$. By Observation 13.17, $[A], [B] \in \mathcal{S}^{V\times V}/_\sim$ are well-defined on $\mathcal{M}^V/_\sim$. Equation (13.31) follows from Equation (13.21)

$$([A] \oplus [B])([x]) = [A]([x]) \oplus [B]([x]) \overset{(13.30)}{=} [A(x)] \oplus [B(x)] = [(A \oplus B)(x)] \tag{13.33}$$
$$\overset{(13.21)}{=} [(A \oplus B)x] \tag{13.34}$$
$$\overset{(13.30)}{=} [A \oplus B][x], \tag{13.35}$$

and Equation (13.32) from Equation (13.22)

$$[A \circ B]([x]) \overset{(13.30)}{=} [(A \circ B)(x)] \overset{(13.22)}{=} [ABx] \overset{(13.30)}{=} [AB][x]. \tag{13.36}$$

It immediately follows from Equations (13.31) and (13.32) that $\mathcal{F}/_\sim$ is a semiring. □

**Observation 13.19.** *$[A] \in \mathcal{F}/_\sim$ is linear.*

*Proof.* Let $s \in \mathcal{S}$ and $x, y \in \mathcal{M}^V$ be arbitrary, compute

$$[A][x \oplus y] = [A(x \oplus y)] = [Ax \oplus Ay] = [Ax] \oplus [Ay] = [A][x] \oplus [A][y] \text{ and} \tag{13.37}$$
$$[A](s[x]) = [A(sx)] = [s(Ax)] = s[Ax] = s[A][x], \tag{13.38}$$

and the claim follows. □

Corollary 13.20 demonstrates that we may leave out or apply additional filter steps whenever convenient. This is a key property which we use throughout the following chapters, e.g., in Chapter 15 where we simulate MBF-like iterations in an implicitly represented graph $H$ whose edges correspond to entire paths in the original graph $G$. Intermediate filtering steps in $G$ keep the intermediate results of the simulation small. Furthermore, as promised in Section 13.3, Corollary 13.20 settles that $(r^V A)^h \sim r^V A^h$. We establish this property separately from Theorem 13.18 because $r^V$ is not an SLF.

**Corollary 13.20** ($r^V \sim \mathrm{id}$)**.** *Let $r$ be a representative projection w.r.t. the congruence relation $\sim$ on the semimodule $\mathcal{M}$ and let $A$ be an SLF. By definition, $r^V$ is well-behaved in terms of Equation* (13.25)*. Moreover, we have $r^V \sim \mathrm{id}$, i.e., it holds that*

$$r^V A \sim A r^V \sim A. \tag{13.39}$$

*In particular, for any MBF-like algorithm $\mathcal{A}$, we have*

$$\mathcal{A}^h(G) \overset{(13.18)}{=} r^V A^h x^{(0)} \overset{(13.39)}{=} (r^V A)^h x^{(0)}. \tag{13.40}$$

Carefully note that, while much of the above machinery may seem quite natural, both the initially chosen restrictions to SLFs and componentwise application of $r$ in $r^V$ are crucial for Corollary 13.20. We begin by pointing out that arbitrary linear functions, as opposed to SLFs, break Corollary 13.20.

**Remark 13.21** (Non-Simple Linear Functions break Corollary 13.20)**.** *Consider $V$, $\mathcal{S}$, $\mathcal{M}$, and $f$ from Remark 13.14. Choose $r(x) := (x_1, \infty)$ for all $x \in \mathcal{M}$. Then*

$$r^V f \begin{pmatrix} (2,1) \\ \perp \end{pmatrix} = r^V \begin{pmatrix} (1,\infty) \\ \perp \end{pmatrix} = \begin{pmatrix} (1,\infty) \\ \perp \end{pmatrix} \text{ but} \tag{13.41}$$

$$f r^V \begin{pmatrix} (2,1) \\ \perp \end{pmatrix} = f \begin{pmatrix} (2,\infty) \\ \perp \end{pmatrix} = \begin{pmatrix} (2,\infty) \\ \perp \end{pmatrix}, \tag{13.42}$$

*implying that $r^V f \not\sim f r^V$.*

Intuitively speaking, it is clear that inconsistencies are introduced when some nodes of a graph run SSSP and others APSP: The nodes running SSSP discard all but the distance to the source node while the nodes running APSP keep all distances. Hence, a node running SSSP may discard vital information about a shortest path. We formally state SSSP and APSP as MBF-like algorithms in Examples 14.4 and 14.6, respectively. In terms of the machinery developed in this chapter, this means that all nodes have to apply the same representative projection, i.e., that $r^V : \mathcal{M}^V \to \mathcal{M}^V$ has to apply $r : \mathcal{M} \to \mathcal{M}$ to all its coordinates:

**Remark 13.22** (Non-componentwise filtering breaks Corollary 13.20)**.** *Consider $V$, $\mathcal{S}$, and $\mathcal{M}$ from Remark 13.14. Suppose $f$ is the SLF given by $f(x) := \binom{x_1 \oplus x_2}{\perp}$ and $r^V \binom{x_1}{x_2} := \binom{x_1}{\perp}$, i.e., $r^V$ applies different representative projections — $\mathrm{id}$ and $\mathrm{const}_\perp$ — to each coordinate of $x \in \mathcal{M}^V$. Then we have that*

$$r^V f \begin{pmatrix} (2,\infty) \\ (1,\infty) \end{pmatrix} = r^V \begin{pmatrix} (1,\infty) \\ \perp \end{pmatrix} = \begin{pmatrix} (1,\infty) \\ \perp \end{pmatrix} \text{ but} \tag{13.43}$$

$$f r^V \begin{pmatrix} (2,\infty) \\ (1,\infty) \end{pmatrix} = f \begin{pmatrix} (2,\infty) \\ \perp \end{pmatrix} = \begin{pmatrix} (2,\infty) \\ \perp \end{pmatrix}, \tag{13.44}$$

*again implying that $r^V f \not\sim f r^V$.*

# CHAPTER 14

# A Collection of MBF-like Algorithms

For the purpose of illustration and to demonstrate the generality of the framework of MBF-like algorithms proposed in Chapter 13, we show that a variety of standard algorithms are MBF-like algorithms; due to the established machinery, this is a trivial task in most cases.

We demonstrate that some more involved distributed algorithms in the CONGEST model have a straightforward and compact interpretation in our framework in Chapter 18. They compute metric tree embeddings based on the FRT distribution; we present them alongside an improved distributed algorithm based on the other results of this work.

MBF-like algorithms are specified by a zero-preserving semimodule $\mathcal{M}$ over a semiring $\mathcal{S}$, a representative projection w.r.t. a congruence relation on $\mathcal{M}$, an initial state $x^{(0)}$, and the number of iterations $h$, compare Definition 13.11. While this is a long list, a standard semiring and semimodule can be chosen; the general-purpose choices of $\mathcal{S} = \mathcal{S}_{\min,+}$ and $\mathcal{M} = \mathcal{D}$ (see Definition 13.1), or $\mathcal{S} = \mathcal{M} = \mathcal{S}_{\min,+}$ (every semiring is a zero-preserving semimodule over itself) often are up to the task. Refer to Sections 14.2–14.4 for examples that require different semirings. However, even in these cases, the semirings and semimodules specified in Sections 14.2–14.4 can be reused. Hence, all that is left to do in most cases is to pick an existing semiring and semimodule, choose $h \in \mathbb{N}_0$, and specify a representative projection $r$.

## 14.1 MBF-like Algorithms over the Min-Plus Semiring

The min-plus semiring — a.k.a. the tropical semiring — is known to be the semiring of choice to capture many standard distance problems. Recall $\mathcal{S}_{\min,+}$ and the adjacency matrix $A \in \mathcal{S}_{\min,+}^{V \times V}$ of the weighted graph $G$, both from Definition 12.3, and the specification of the distance-map semimodule $\mathcal{D}$ from Definition 13.1. In that light consider the initialization $x^{(0)} \in \mathcal{D}^V$ with

$$x^{(0)}_{vw} := \begin{cases} 0 & \text{if } v = w \text{ and} \\ \infty & \text{otherwise} \end{cases} \tag{14.1}$$

and observe that the entries of

$$x^{(h)} := A^h x^{(0)} \tag{14.2}$$

correspond to the $h$-hop distances in $G$. It is well-known that the min-plus semiring can be used for distance computations [5, 35, 115, 135]. Nevertheless, for the sake of completeness, we prove the following lemma in terms of our notation and in the light of the framework of MBF-like algorithms proposed in Chapter 13.

**Lemma 14.1.** *For $h \in \mathbb{N}_0$ and $x^{(h)}$ from Equation* (14.2)*, we have*

$$x^{(h)}_{vw} = \operatorname{dist}^h(v, w, G). \tag{14.3}$$

*Proof.* The claim trivially holds for $h = 0$. As induction hypothesis, suppose the claim holds for $h \in \mathbb{N}_0$. We obtain

$$\begin{aligned}
x_{vw}^{(h+1)} &= \left(Ax^{(h)}\right)_{vw} && (14.4)\\
&= \left(\bigoplus_{u \in V} a_{vu} \odot x_u^{(h)}\right)_w && (14.5)\\
&= \bigoplus_{u \in V} a_{vu} \odot x_{uw}^{(h)} && (14.6)\\
&= \min_{u \in V} \left\{a_{vu} + x_{uw}^{(h)}\right\} && (14.7)\\
&= \min \left\{\omega(v,u) + \operatorname{dist}^h(u,w,G) \mid \{v,u\} \in E\right\} \cup \left\{0 + \operatorname{dist}^h(v,w,G)\right\}, && (14.8)
\end{aligned}$$

i.e., exactly the definition of $\operatorname{dist}^{h+1}(v,w,G)$, as claimed. □

As a first example, we turn our attention to source detection. It generalizes all examples covered in this section, saving us from proving each of them correct individually; well-established examples like SSSP and APSP follow. Source detection was introduced by Lenzen and Peleg [104]; we extend their definition by a maximum distance $d$.

**Definition 14.2** (Source Detection [104])**.** *Given a weighted graph $G = (V, E, \omega)$, sources $S \subseteq V$, hop and result limits $h, k \in \mathbb{N}_0$, and a maximum distance $d \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, $(S, h, d, k)$-*source detection *is the following problem: For each $v \in V$, determine the $k$ smallest elements of $\{(\operatorname{dist}^h(v,s,G), s) \mid s \in S, \operatorname{dist}(v,s,G) \leq d\}$ w.r.t. lexicographical order; determine all of them if there are fewer than $k$.*

**Example 14.3** (Source Detection)**.** *Source detection is solved by $h$ iterations of an MBF-like algorithm with $\mathcal{S} = \mathcal{S}_{\min,+}$, $\mathcal{M} = \mathcal{D}$,*

$$r(x)_v = \begin{cases} x_v & \text{if } v \in S,\ x_v \leq d \text{ and } x_v \text{ is among } k \text{ smallest } S\text{-entries of } x \\ & \quad \text{(ties broken by index) and} \\ \infty & \text{otherwise,} \end{cases} \qquad (14.9)$$

*and $x^{(0)}$ from Equation* (14.1)*.*

*Proof.* Let $s \in \mathcal{S}_{\min,+}$ be arbitrary and let $x, x', y, y' \in \mathcal{D}$ be such that $x \sim x'$ and $y \sim y'$, where $x \sim y :\Leftrightarrow r(x) = r(y)$. By Lemma 13.8, it suffices to show that (1) $r^2 = r$, (2) $r(sx) = r(sx')$, and (3) $r(x \oplus y) = r(x' \oplus y')$. We show the claims one by one.

(1) $r(x)$ has at most $k$ non-$\infty$ entries, each at most $d$, so $r(r(x)) = r(x)$ by (14.9).

(2) Since multiplication with $s$ uniformly increases the non-$\infty$ entries of $x$ and $x'$, it does not affect their order w.r.t. (14.9). As the $k$ smallest $S$-entries of $x$ and $x'$ w.r.t. (14.9) are identical (the same nodes and the same values), so are those of $sx$ and $sx'$. Some entry $(sx)_v$ may become larger than $d$ (if $x_v \leq d < s + x_v$), but that happens if and only if it does for $(sx')_v$ as well, hence $r(sx) = r(sx')$.

(3) We show that $r(x \oplus y) = r(r(x) \oplus r(y))$; the claim then follows from observing that $r(x \oplus y) = r(r(x) \oplus r(y)) = r(r(x') \oplus r(y')) = r(x' \oplus y')$.

   To see that $r(x \oplus y) = r(r(x) \oplus r(y))$ we show that, for all $v \in V$, we either have $(x \oplus y)_v = (r(x) \oplus r(y))_v$ or that $r(x \oplus y)_v = r(r(x) \oplus r(y))_v = \infty$, i.e., the entries of $x \oplus y$ and $r(x) \oplus r(y)$ are either equal or set to $\infty$ by $r$. Suppose that $(x \oplus y)_v \neq (r(x) \oplus r(y))_v$ for some $v \in V$. It follows that $(x \oplus y)_v < (r(x) \oplus r(y))_v = \infty$ as well as $v \in S$ because $r$ only modifies $S$-entries by setting them to $\infty$. As $(r(x) \oplus r(y))_v = \min\{r(x)_v, r(y)_v\}$, it follows that $x_v$ is not among the $k$ smallest $S$-entries of $x$ and that $y_v$ is not among the $k$ smallest $S$-entries of $y$. Hence, $v$ cannot be among the $k$ smallest $S$-entries of $x \oplus y$. It follows that $r(x \oplus y)_v = r(r(x) \oplus r(y))_v = \infty$.

Together, this fulfills the conditions of Lemma 13.8 and the claim follows. □

**Example 14.4** (Single-Source Shortest Paths)**.** Single-Source Shortest Paths (SSSP) *requires to determine the h-hop distance to* $s \in V$ *for all* $v \in V$*. It is solved by an MBF-like algorithm with* $\mathcal{S} = \mathcal{M} = \mathcal{S}_{\min,+}$*,* $r = \mathrm{id}$*, and* $x_s^{(0)} = 0$ *and* $x_v^{(0)} = \infty$ *for all* $v \neq s$*.*

Equivalently, one may use $(\{s\}, h, \infty, 1)$-source detection. This effectively results in $\mathcal{M} = \mathcal{S}_{\min,+}$: when only storing the non-$\infty$ entries, only the $s$-entry is relevant.

**Example 14.5** ($k$-Source Shortest Paths)**.** $k$-Source Shortest Paths ($k$-SSP) *requires to determine, for each node, the k closest nodes in terms of the h-hop distance* $\mathrm{dist}^h(\cdot,\cdot,G)$*. It is solved by an MBF-like algorithm, as it corresponds to* $(V, h, \infty, k)$*-source detection.*

**Example 14.6** (All-Pairs Shortest Paths)**.** All-Pairs Shortest Paths (APSP) *is the task of determining the h-hop distance between all pairs of nodes. It is solved by an MBF-like algorithm because we can use* $(V, h, \infty, n)$*-source detection, resulting in* $\mathcal{M} = \mathcal{D}$*,* $r = \mathrm{id}$*, and* $x^{(0)}$ *from Equation* (14.1)*.*

**Example 14.7** (Multi-Source Shortest Paths)**.** *The* Multi-Source Shortest Paths (MSSP) *problem is to determine, for each node, the h-hop distances to all nodes in a designated set* $S \subseteq V$ *of source nodes. This is solved by the MBF-like algorithm for* $(S, h, \infty, |S|)$*-source detection.*

**Example 14.8** (Forest Fires)**.** *The nodes in a graph G form a distributed sensor network, the edges represent communication channels, and edge weights correspond to distances. Our goal is to detect, for each node v, if there is a node w on fire within distance* $\mathrm{dist}(v, w, G) \leq d$ *for some* $d \in \mathbb{R}_{\geq 0} \cup \{\infty\}$*, where every node initially knows if it is on fire. As a suitable MBF-like algorithm, pick* $h = n$*,* $\mathcal{S} = \mathcal{M} = \mathcal{S}_{\min,+}$*,*

$$r(x) = \begin{cases} x & \text{if } x \leq d \text{ and} \\ \infty & \text{otherwise,} \end{cases} \tag{14.10}$$

*and* $x_v^{(0)} = 0$ *if v is on fire and* $x_v^{(0)} = \infty$ *otherwise.*

Example 14.8 can be handled differently by using $(S, n, d, 1)$-source detection, where $S$ are the nodes on fire. This also reveals the closest node on fire, whereas the solution from Example 14.8 is anonymous. One can interpret both solutions as instances of SSSP with a virtual source $s \notin V$ that is connected to all nodes on fire by an edge of weight 0. This, however, requires a simulation argument and additional reasoning if the closest node on fire is to be determined.

## 14.2 MBF-like Algorithms over the Max-Min Semiring

Some problems require using a semiring other than $\mathcal{S}_{\min,+}$. As a first example, consider the *Widest Path Problem (WPP),* also referred to as the bottleneck shortest path problem: Given two nodes $v$ and $w$ in a weighted graph, find a $v$-$w$-path maximizing the lightest edge in the path. More formally, we are interested in the widest-path distance between $v$ and $w$:

**Definition 14.9** (Widest-Path Distance)**.** *Given a weighted graph $G = (V, E, \omega)$, a path $p$ has* width $\operatorname{width}(p) := \min\{\omega(e) \mid e \in p\}$*. The $h$-*hop widest-path distance *between $v, w \in V$ is*

$$\operatorname{width}^h(v, w, G) := \max\left\{\operatorname{width}(p) \mid p \in \mathrm{P}^h(v, w, G)\right\}. \tag{14.11}$$

*We abbreviate* $\operatorname{width}(v, w, G) := \operatorname{width}^n(v, w, G)$.

An application of the WPP are trust networks: The nodes of a graph are entities and an edge $\{v, w\}$ of weight $0 < \omega(v, w)$ encodes that $v$ and $w$ trust each other with $\omega(v, w)$. Assuming trust to be transitive, $v$ trusts $w$ with $\max_{p \in \mathrm{P}(v,w,G)} \min_{e \in p} \omega(e) = \operatorname{width}(v, w, G)$. The WPP requires a semiring supporting the max and min operations:

**Definition 14.10** (Max-Min Semiring)**.** *We refer to* $\mathcal{S}_{\max,\min} := (\mathbb{R}_{\geq 0} \cup \{\infty\}, \max, \min)$ *as the* max-min semiring.

For the sake of completeness, we prove the following standard result.

**Lemma 14.11.** *$\mathcal{S}_{\max,\min}$ is a semiring with neutral elements $0$ and $\infty$.*

*Proof.* We check each of the requirements of Definition 2.2. Throughout the proof, let $x, y, z \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ be arbitrary.

(1) $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \max)$ is a commutative semigroup because max is associative and commutative. Since 0 is the minimum of $\mathbb{R}_{\geq 0} \cup \{\infty\}$, it is the neutral element of $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \max)$.

(2) $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \min)$ is a semigroup because min is associative. Its neutral element is $\infty$ because it is the maximum of $\mathbb{R}_{\geq 0} \cup \{\infty\}$.

(3) Regarding the left- and right-distributive laws in Equations (2.4) and (2.5), a case distinction between the cases (a) $x \leq y \leq z$, (b) $y \leq x \leq z$, and (c) $y \leq z \leq x$ is exhaustive due to the commutativity of max. It reveals that

$$\min\{x, \max\{y, z\}\} = \max\{\min\{x, y\}, \min\{x, z\}\}, \tag{14.12}$$

i.e., that the left-distributive law holds. Since min is commutative,

$$\min\{\max\{y,z\},x\} = \max\{\min\{y,x\},\min\{z,x\}\} \tag{14.13}$$

immediately follows from Equation (14.12).

(4) 0 is an annihilator for min because

$$\min\{0,x\} = \min\{x,0\} = 0 \tag{14.14}$$

for all $x \in \mathbb{R}_{\geq 0} \cup \{\infty\}$.

Together, it follows that $\mathcal{S}_{\max,\min}$ is a semiring as claimed. □

**Corollary 14.12.** *$\mathcal{S}_{\max,\min}$ is a zero-preserving semimodule over itself. Furthermore, we have that $\mathcal{W} := \mathcal{S}^V_{\max,\min}$ is a zero-preserving semimodule over $\mathcal{S}_{\max,\min}$ by Lemma 2.4.*

As adjacency matrix of $G = (V, E, \omega)$ w.r.t. $\mathcal{S}_{\max,\min}$ we propose $A \in \mathcal{S}^{V\times V}_{\max,\min}$ with

$$(a_{vw}) := \begin{cases} \infty & \text{if } v = w, \\ \omega(v,w) & \text{if } \{v,w\} \in E, \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{14.15}$$

This is a straightforward adaptation of the adjacency matrix w.r.t. $\mathcal{S}_{\min,+}$ in Equation (12.17). As an initialization $x^{(0)} \in \mathcal{W}^V$ in which each node knows the trivial path of unbounded width to itself but nothing else is given by

$$x^{(0)}_{vw} := \begin{cases} \infty & \text{if } v = w \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{14.16}$$

Then $h \in \mathbb{N}_0$ multiplications with $A$, i.e., $h$ iterations, yield

$$x^{(h)} := A^h x^{(0)} \tag{14.17}$$

which corresponds to the $h$-hop widest-path distance:

**Lemma 14.13.** *Given $x^{(h)}$ from Equation* (14.17)*, we have*

$$x^{(h)}_{vw} = \operatorname{width}^h(v,w,G). \tag{14.18}$$

*Proof.* The claim holds for $h = 0$ by Equation (14.16). As induction hypothesis, suppose the claim holds for some $h \in \mathbb{N}_0$. We obtain

$$x^{(h+1)}_v \overset{(14.17)}{=} \left(Ax^{(h)}\right)_v = \bigoplus_{w\in V} a_{vw} \odot x^{(h)}_w \overset{(14.15)}{=} \underbrace{\infty \odot x^{(h)}_v}_{x^{(h)}_v} \oplus \bigoplus_{\{v,w\}\in E} \omega(v,w) \odot x^{(h)}_w. \tag{14.19}$$

As $\oplus$ in $\mathcal{W}$ is the element-wise maximum, we have

$$x^{(h+1)}_{vu} = \max\left\{x^{(h)}_{vu}\right\} \cup \left\{\min\left\{\omega(v,w), x^{(h)}_{wu}\right\} \mid \{v,w\} \in E\right\} \tag{14.20}$$

and the induction hypothesis yields

$$\begin{aligned} x_{vu}^{(h+1)} = \max & \left\{ \operatorname{width}^h(v,u,G) \right\} \\ & \cup \left\{ \min \left\{ \omega(v,w), \operatorname{width}^h(w,u,G) \right\} \mid \{v,w\} \in E \right\}, \end{aligned} \tag{14.21}$$

which is exactly $\operatorname{width}^{h+1}(v,u,G)$. □

**Example 14.14** (Single-Source Widest Paths)**.** Single-Source Widest Paths (SSWP) *asks for, given a weighted graph $G = (V, E, \omega)$, a designated source node $s \in V$, and $h \in \mathbb{N}_0$, the h-hop widest-path distance* $\operatorname{width}^h(s,v,G)$ *for every $v \in V$. It is solved by an MBF-like algorithm with $\mathcal{S} = \mathcal{M} = \mathcal{S}_{\max,\min}$, $r = \mathrm{id}$, and $x_s^{(0)} = \infty$ and $x_v^{(0)} = 0$ for all $v \neq s$.*

**Example 14.15** (All-Pairs Widest Paths)**.** All-Pairs Widest Paths (APWP) *asks for, given $G = (V, E, \omega)$ and $h \in \mathbb{N}_0$,* $\operatorname{width}^h(v,w,G)$ *for all $v, w \in V$. APWP is MBF-like; it is solved by choosing $\mathcal{S} = \mathcal{S}_{\max,\min}$, $\mathcal{M} = \mathcal{W}$, $r = \mathrm{id}$, and $x^{(0)}$ from Equation* (14.16) *by Lemma 14.13.*

**Example 14.16** (Multi-Source Widest Paths)**.** *The* Multi-Source Widest Paths (MSWP) *problem is to determine, for each node, the h-hop widest path distance to all nodes in a designated set $S \subseteq V$ of source nodes. This is solved by the same MBF-like algorithm as for APWP in Example 14.15 when changing $x^{(0)}$ to $x_{vw}^{(0)} = \infty$ if $v = w \in S$ and $x_{vw}^{(0)} = 0$ otherwise.*

## 14.3 MBF-like Algorithms over the All-Paths Semiring

Mohri discusses the $k$-SDP, where each $v \in V$ is required to find the $k$ shortest paths to a designated source node $s \in V$, in the light of his algebraic framework for distance computations [115]. Our framework captures this application as well, but requires a different semiring than $\mathcal{S}_{\min,+}$: While $\mathcal{S}_{\min,+}$ suffices for many applications, see Section 14.1, it cannot distinguish between different paths of the same length. This is a problem in the $k$-SDP, because there may be multiple paths of the same length among the $k$ shortest.

**Observation 14.17.** *No semimodule $\mathcal{M}$ over $\mathcal{S}_{\min,+}$ can overcome this issue: The left-distributive law for semimodules, see Equation* (2.9)*, requires, for all $x \in \mathcal{M}$ and $s, s' \in \mathcal{S}_{\min,+}$, that $sx \oplus s'x = (s \oplus s')x$. Consider different paths $\pi \neq \pi'$ ending in the same node with $\omega(\pi) = s = s' = \omega(\pi')$. W.r.t. $\mathcal{S}_{\min,+}$ and $\mathcal{M}$, the left-distributive law yields $sx \oplus s'x = \min\{s, s'\} \odot x$, i.e., propagating $x$ over $\pi$, over $\pi'$, or over both and then aggregating* must be indistinguishable *in the case of $s = s'$.*

This does not mean that the framework of MBF-like algorithms cannot be applied; it merely indicates that we need a more powerful semiring than $\mathcal{S}_{\min,+}$. The motivation of this section is to generalize $\mathcal{S}_{\min,+}$ to the *all-paths semiring* $\mathcal{P}_{\min,+}$ and add it to the toolbox. With $\mathcal{P}_{\min,+}$ in place, the established machinery becomes available: pick a semimodule (in our examples, $\mathcal{P}_{\min,+}$ itself suffices) and define a representative projection. We demonstrate this for the $k$-SDP and a variant.

The basic concept of $\mathcal{P}_{\min,+}$ is to remember *paths* instead of destination–distance pairs. Furthermore, we include the ability to remember multiple paths, regardless of whether they have the same weight. This includes enough features in $\mathcal{P}_{\min,+}$; we do not require dedicated semimodules for $k$-SDP and use the fact that $\mathcal{P}_{\min,+}$ is a zero-preserving semimodule over itself.

We begin with a convenient representation of paths: Let $P = V^+$ denote the set of directed paths on $V$, with loops, denoted as non-empty tuples of nodes. Furthermore, let $\circ \subseteq P^2$ be the relation of *concatenable* paths defined by

$$(v_1, \dots, v_k) \circ (w_1, \dots, w_\ell) \quad :\Leftrightarrow \quad v_k = w_1. \tag{14.22}$$

By abuse of notation, we occasionally use $\circ$ as concatenation operator when and if its operands are concatenable. Furthermore, we abbreviate the rather cumbersome $\{(\pi^1, \pi^2) \in P^2 \mid \pi^1 \circ \pi^2 \wedge \pi \text{ is the concatenation of } \pi^1 \text{ and } \pi^2\}$ by the more compact $\{(\pi^1, \pi^2) \mid \pi = \pi^1 \circ \pi^2\}$ to denote all two-splits of $\pi$.

Let $\pi = (v_1, \dots, v_k) \in P$ be a sequence of $k \in \mathbb{N}$ nodes. In the context of a graph $G = (V, E, \omega)$, we generalize $\omega(\pi) := \sum_{i=1}^{k-1} \omega(v_i, v_{i+1})$. We call $\pi$ *valid* if and only if $\omega(\pi) \neq \infty$ and *invalid* otherwise (recall that $\omega(v, w) = \infty$ if $\{v, w\} \notin E$). Observe that $\pi$ uniquely determines a sequence of $k - 1$ edges and is valid if and only if all of these edges exist in $E$. Furthermore, recall that $\emptyset \in \mathrm{P}^0(v, v, G)$ for all $v \in V$ and observe that we consider single-node paths $(v)$ valid paths of weight 0. Together, all $p \in \mathrm{P}(\cdot, \cdot, G)$ have a uniquely determined, valid, vertex-based representation $\pi \in P$ with $\omega(p) = \omega(\pi)$.

As motivated above, the all-paths semiring is a generalization of $\mathcal{S}_{\min,+}$ capable of remembering multiple paths. We represent this using vectors in $(\mathbb{R}_{\geq 0} \cup \{\infty\})^P$ storing a non-$\infty$ weight for each explored path and $\infty$ for all others. Making the $\infty$ entries implicit creates an efficient representation.

**Definition 14.18** (All-Paths Semiring)**.** *We call* $\mathcal{P}_{\min,+} = ((\mathbb{R}_{\geq 0} \cup \{\infty\})^P, \oplus, \odot)$ *the* all-paths semiring, *where* $\oplus$ *and* $\odot$ *are defined, for all* $\pi \in P$ *and* $x, y \in \mathcal{P}_{\min,+}$, *by*

$$(x \oplus y)_\pi := \min\{x_\pi, y_\pi\} \text{ and} \tag{14.23}$$

$$(x \odot y)_\pi := \min\{x_{\pi^1} + y_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\}. \tag{14.24}$$

*We say that* $x$ contains $\pi$ (with weight $x_\pi$) *if and only if* $x_\pi < \infty$.

Summation $(x \oplus y)_\pi$ picks the smallest weight associated to the path $\pi$ in either operand. Multiplication $(x \odot y)_\pi$ finds the lightest estimate for $\pi$ composed of two-splits $\pi = \pi^1 \circ \pi^2$, where $\pi^1$ is picked from $x$ and $\pi^2$ from $y$. Note carefully that $\pi^1$ and $\pi^2$ have to contain at least one node by definition of $P$, hence, $y$ may contain paths not contained $x \odot y$. Observe that $\mathcal{P}_{\min,+}$ supports upper bounds on path lengths; we do, however, not use this feature. Intuitively, $\mathcal{P}_{\min,+}$ stores all encountered paths with their exact weights; in this mindset, summation corresponds to the union and multiplication to the "concatenability-obeying Cartesian product" of the paths contained in $x$ and $y$.

**Lemma 14.19.** $\mathcal{P}_{\min,+}$ *is a semiring with neutral elements*

$$0 := (\infty, \dots, \infty)^\top, \text{ and} \tag{14.25}$$

$$1_\pi := \begin{cases} 0 & \text{if } \pi = (v) \text{ for some } v \in V \text{ and} \\ \infty & \text{otherwise} \end{cases} \tag{14.26}$$

*w.r.t.* $\oplus$ *and* $\odot$*, respectively.*

*Proof.* We check the requirements of Definition 2.2 step by step. Throughout the proof, let $\pi \in P$ and $x, y, z \in \mathcal{P}_{\min,+}$ be arbitrary.

(1) We first show that $((\mathbb{R}_{\geq 0} \cup \{\infty\})^P, \oplus)$ is a commutative semigroup with neutral element 0. The associativity of $\oplus$ — and with it the property of $((\mathbb{R}_{\geq 0} \cup \{\infty\})^P, \oplus)$ being a semigroup — follows from the associativity of min:

$$\begin{aligned} ((x \oplus y) \oplus z)_\pi &= \min\{\min\{x_\pi, y_\pi\}, z_\pi\} && (14.27)\\ &= \min\{x_\pi, \min\{y_\pi, z_\pi\}\} && (14.28)\\ &= (x \oplus (y \oplus z))_\pi. && (14.29)\end{aligned}$$

$\oplus$ is commutative, because min is. It is easy to check that $(x \oplus 0)_\pi = (0 \oplus x)_\pi = x_\pi$.

(2) To see that $((\mathbb{R}_{\geq 0} \cup \{\infty\})^P, \odot)$ is a semigroup with neutral element 1, we first check that $\odot$ is associative, i.e., that it is a semigroup:

$$\begin{aligned} ((x \odot y) \odot z)_\pi &= \min\{(x_{\pi^1} + y_{\pi^2}) + z_{\pi^3} \mid \pi = (\pi^1 \circ \pi^2) \circ \pi^3\} && (14.30)\\ &= \min\{x_{\pi^1} + (y_{\pi^2} + z_{\pi^3}) \mid \pi = \pi^1 \circ (\pi^2 \circ \pi^3)\} && (14.31)\\ &= (x \odot (y \odot z))_\pi. && (14.32)\end{aligned}$$

Furthermore, $(1 \odot x)_\pi = \min\{0 + x_\pi\} = x_\pi = (x \odot 1)_\pi$, hence 1 is the neutral element w.r.t. $\odot$.

(3) Regarding the distributive laws, we begin with the left-distributive law (2.4):

$$\begin{aligned} (x \odot (y \oplus z))_\pi &= \min\{x_{\pi^1} + \min\{y_{\pi^2}, z_{\pi^2}\} \mid \pi = \pi^1 \circ \pi^2\} && (14.33)\\ &= \min\{\min\{x_{\pi^1} + y_{\pi^2}, x_{\pi^1} + z_{\pi^2}\} \mid \pi = \pi^1 \circ \pi^2\} && (14.34)\\ &= \min\left\{\begin{matrix}\min\{x_{\pi^1} + y_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\},\\ \min\{x_{\pi^1} + z_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\}\end{matrix}\right\} && (14.35)\\ &= ((x \odot y) \oplus (x \odot z))_\pi. && (14.36)\end{aligned}$$

Regarding the right-distributive law (2.5), we obtain:

$$\begin{aligned} ((y \oplus z) \odot x)_\pi &= \min\{\min\{y_{\pi^1}, z_{\pi^1}\} + x_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\} && (14.37)\\ &= \min\{\min\{y_{\pi^1} + x_{\pi^2}, z_{\pi^1} + x_{\pi^2}\} \mid \pi = \pi^1 \circ \pi^2\} && (14.38)\\ &= \min\left\{\begin{matrix}\min\{y_{\pi^1} + x_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\},\\ \min\{z_{\pi^1} + x_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\}\end{matrix}\right\} && (14.39)\\ &= ((y \odot x) \oplus (z \odot x))_\pi. && (14.40)\end{aligned}$$

(4) It remains to check that 0 is an annihilator w.r.t. $\odot$. We have

$$\begin{aligned} (0 \odot x)_\pi &= \min\{0_{\pi^1} + x_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\} && (14.41)\\ &= \min\{\infty + x_{\pi^2} \mid \pi = \pi^1 \circ \pi^2\} && (14.42)\\ &= \infty && (14.43)\\ &= 0_\pi && (14.44)\end{aligned}$$

and, analogously, $(x \odot 0)_\pi = 0_\pi$.

Together, $\mathcal{P}_{\min,+}$ is a semiring as claimed. □

**Corollary 14.20.** *$\mathcal{P}_{\min,+}$ is a zero-preserving semimodule over itself.*

Computations in a graph $G = (V, E, \omega)$ w.r.t. $\mathcal{P}_{\min,+}$ require an adjacency matrix $A \in \mathcal{P}_{\min,+}^{V \times V}$. We propose the following generalization of Equation (12.17) with

$$(a_{vw})_\pi := \begin{cases} 1_\pi & \text{if } v = w, \\ \omega(v,w) & \text{if } \pi = (v,w), \text{ and} \\ \infty & \text{otherwise.} \end{cases} \tag{14.45}$$

On the diagonal, $a_{vv} = 1$ contains exactly the zero-hop paths of weight 0; all non-trivial paths are "unknown" in $a_{vv}$, i.e., accounted for with an infinite weight. An entry $a_{vw}$ with $v \neq w$ contains, if present, only the edge $\{v, w\}$, represented by the path $(v, w)$ of weight $\omega(v, w)$; no other path is contained in $a_{vw}$. An initialization where each node $v$ knows only the zero-hop path $(v)$ is represented by the vector $x^{(0)} \in \mathcal{P}_{\min,+}^{V}$ with

$$\left(x_v^{(0)}\right)_\pi := \begin{cases} 0 & \text{if } \pi = (v) \text{ and} \\ \infty & \text{otherwise.} \end{cases} \tag{14.46}$$

Then $h \in \mathbb{N}_0$ multiplications of $x^{(0)}$ with $A$, i.e., $h$ iterations, yield

$$x^{(h)} := A^h x^{(0)} \tag{14.47}$$

and $x_v^{(h)}$ contains exactly the $h$-hop paths beginning in $v$ with their according weights:

**Lemma 14.21.** *Let $x^{(h)}$ be defined as in Equation* (14.46)*, w.r.t. the graph $G = (V, E, \omega)$. Then for all $v \in V$ and $\pi \in P$*

$$\left(x_v^{(h)}\right)_\pi = \begin{cases} \omega(\pi) & \textit{if } \pi \in \mathrm{P}^h(v, \cdot, G) \textit{ and} \\ \infty & \textit{otherwise.} \end{cases} \tag{14.48}$$

*Proof.* We prove the claim by induction. The claim holds for $h = 0$ by Equation (14.46). As induction hypothesis, suppose the claim holds for $h \in \mathbb{N}_0$. The induction step yields

$$x_v^{(h+1)} \overset{(14.47)}{=} \left(Ax^{(h)}\right)_v = \bigoplus_{w \in V} a_{vw} x_w^{(h)} \overset{(14.45)}{=} \underbrace{a_{vv}}_{1} x_v^{(h)} \oplus \bigoplus_{\{v,w\} \in E} a_{vw} x_w^{(h)}. \tag{14.49}$$

We have $a_{vv} x_v^{(h)} = 1 x_v^{(h)} = x_v^{(h)}$ by construction, i.e., $a_{vv} x_v^{(h)}$ contains exactly the properly weighted $h$-hop paths beginning at $v$ by the induction hypothesis. Next, consider $\{v, w\} \in E$. By induction, $x_w^{(h)}$ contains exactly the $h$-hop paths beginning in $w$ and $a_{vw}$ contains only the edge $\{v, w\}$ of weight $\omega(v, w)$ by Equation (14.45). Hence, $a_{vw} x^{(h)}$ contains all $(h+1)$-hop paths beginning with $\{v, w\}$. Due to Equation (14.49) and to

$$\mathrm{P}^{h+1}(v, \cdot, G) = \mathrm{P}^h(v, \cdot, G) \ \cup \bigcup_{\{v,w\} \in E} \left\{(v, w) \circ \pi \mid \pi \in \mathrm{P}^h(w, \cdot, G)\right\}, \tag{14.50}$$

$x_v^{(h+1)}$ contains exactly the properly weighted $(h+1)$-hop paths, as claimed. □

With the all-paths semiring $\mathcal{P}_{\min,+}$ established, we turn to the $k$-SDP, our motivation for adding $\mathcal{P}_{\min,+}$ to the toolbox of MBF-like algorithms in the first place.

**Definition 14.22** ($k$-Shortest Distance Problem [115])**.** *Given a graph $G = (V, W, \omega)$ and a designated source vertex $s \in V$, the* $k$-Shortest Distance Problem ($k$-SDP) *asks: For each node $v \in V$ and considering all $v$-$s$-paths, what are the weights of the $k$ lightest such paths? In the* $k$-Distinct-Shortest Distance Problem ($k$-DSDP), *the path weights have to be distinct.*

With $\mathcal{S} = \mathcal{M} = \mathcal{P}_{\min,+}$ and $r = \mathrm{id}$, we already have an inefficient solution for the $k$-SDP and the $k$-DSDP in place: This corresponds to collecting all $h$-hop paths and then picking only $k$ at each node. In terms of efficiency, however, this is like solving SSSP by first solving APSP and then discarding almost all distances. In order to solve the $k$-SDP more efficiently, we require a representative projection that reduces the abundance of paths stored in an unfiltered $x^{(h)}$ to the relevant ones.

Observe that with the above definitions of $A$ and $x^{(h)}$, we always associate a path $\pi$ with either its weight $\omega(\pi)$ or with $\infty$; in particular, invalid paths always are associated with $\infty$. Formally, $G$ induces a subsemiring of $\mathcal{P}_{\min,+}$. Besides being an interesting observation, these properties are required for the representative projections defined below — $r$ breaks for the $k$-DSDP when facing inconsistent non-$\infty$ values for the same path — so we formalize them. Let $G = (V, E, \omega)$ a graph and let $D(G) \subset (\mathbb{R}_{\geq 0} \cup \{\infty\})^P$ be the restriction of $(\mathbb{R}_{\geq 0} \cup \{\infty\})^P$ to exact path weights and $\infty$:

$$x_\pi \in \{\omega(\pi), \infty\}; \tag{14.51}$$

recall that $\omega(\pi) = \infty$ for all paths $\pi$ invalid w.r.t. $G$.

**Definition 14.23** (Graph-Induced All-Paths Semiring)**.** *Let $G$ be a graph and $D(G) \subset (\mathbb{R}_{\geq 0} \cup \{\infty\})^P$ as above. Then we refer to $\mathcal{P}_{\min,+}(G) := (D(G), \oplus, \odot)$ as the* all-paths semiring induced by $G$, *where $\oplus$ and $\odot$ are the same as in Definition 14.18.*

The next step is to show that $\mathcal{P}_{\min,+}(G)$ is a semiring. It actually is a subsemiring of $\mathcal{P}_{\min,+}$. We do, however, not use that property and hence refrain from formally introducing the according definitions.

**Lemma 14.24.** *$\mathcal{P}_{\min,+}(G)$ is a semiring.*

*Proof.* Fix a graph $G = (V, E, \omega)$. We show that $D(G)$ is closed under $\oplus$ and $\odot$. Consider $x, y \in D(G)$ and let $\pi \in P$ be a path. Hence, we have $x_\pi, y_\pi \in \{\omega(\pi), \infty\}$ if $\pi$ is valid and $x_\pi = y_\pi = \infty$ if $\pi$ is invalid in $G$; recall that we defined $\omega(\pi) = \infty$ for invalid paths.

(1) Consider $x \oplus y$. It directly follows from $x_\pi, y_\pi \in \{\omega(\pi), \infty\}$ that $(x \oplus y)_\pi = \min\{x_\pi, y_\pi\} \in \{\omega(\pi), \infty\}$. Hence, $(x \oplus y) \in D(G)$.

(2) Regarding multiplication, we have $(x \odot y)_\pi = x_{\pi^1} + y_{\pi^2}$ for some two-split of $\pi$. Due to $x_{\pi^1} \in \{\omega(\pi^1), \infty\}$ and $y_{\pi^2} \in \{\omega(\pi^2), \infty\}$, we obtain $(x \odot y)_\pi = x_{\pi^1} + y_{\pi^2} \in \{\omega(\pi^1) + \omega(\pi^2), \infty\} = \{\omega(\pi), \infty\}$. If $\pi$ is invalid in $G$, $\pi^1$ or $\pi^2$ must be invalid and $(x \odot y)_\pi = x_{\pi^1} + y_{\pi^2} = \infty$ follows.

Also observe that $0, 1 \in D(G)$ for $0$ and $1$ from Lemma 14.19. Together, it follows from Lemma 14.19 that $\mathcal{P}_{\min,+}(G)$ is a semiring. $\square$

**Corollary 14.25.** *$\mathcal{P}_{\min,+}(G)$ is a zero-preserving semimodule over itself.*

Observe that we have $A \in \mathcal{P}_{\min,+}^{V \times V}(G)$ as well as $x^{(0)} \in \mathcal{P}_{\min,+}^{V}(G)$. It follows that Lemma 14.21 holds for $\mathcal{P}_{\min,+}(G)$ as much as it does for $\mathcal{P}_{\min,+}$. Furthermore, observe that the restriction to $\mathcal{P}_{\min,+}(G)$ happens implicitly, simply by starting with the above initialization. There is no information about $\mathcal{P}_{\min,+}(G)$ that needs to be distributed in the graph in order to run an MBF-like algorithm over $\mathcal{P}_{\min,+}(G)$.

With $\mathcal{P}_{\min,+}(G)$ in place, we turn our attention back to the $k$-SDP. Specifically, we need a representative projection that, given the abundance of possible paths collected by the all-paths semimodule, discards all but the relevant paths. Relevant in this case means to keep the $k$ shortest $v$-$s$-paths. In order to formalize this, let $P(v, w, x)$ denote, for $x \in \mathcal{P}_{\min,+}(G)$ and $v, w \in V$, the set of $v$-$w$-paths contained in $x$:

$$P(v, w, x) := \{\pi \in P \mid \pi \text{ is a } v\text{-}w\text{-path with } x_\pi \neq \infty\}. \tag{14.52}$$

Order $P(v, w, x)$ ascendingly w.r.t. the weights $x_\pi$, breaking ties using lexicographical order on $P$. Then let $P_k(v, w, x)$ denote the set of the first at most $k$ entries of that sequence:

$$P_k(v, w, x) := \{\pi \mid (x_\pi, \pi) \text{ is among } k \text{ smallest of } \{(x_{\pi'}, \pi') \mid \pi' \in P(v, w, x)\}\}. \tag{14.53}$$

We define the (representative, see below) projection $r \colon \mathcal{P}_{\min,+}(G) \to \mathcal{P}_{\min,+}(G)$ by

$$r(x)_\pi := \begin{cases} x_\pi & \text{if } \pi \in P_k(v, s, x) \text{ for some } v \in V \text{ and} \\ \infty & \text{otherwise.} \end{cases} \tag{14.54}$$

If $x_\pi = r(x)_\pi$ we say that $r$ *keeps* $\pi$ and otherwise that $r$ *discards* $\pi$. The projection $r$ keeps, for each $v \in V$, exactly the $k$ shortest $v$-$s$-paths contained in $x$. Let $\pi$ be a $v$-$w$-path and $y \in \mathcal{P}_{\min,+}(G)$ such that $y$ contains only $\pi$, i.e., with $y_\pi = \omega(\pi)$ and $y_{\pi'} = \infty$ for all $\pi' \neq \pi$. We say that $\pi$ *is dominated in* $x$ if and only if $r(x \oplus y)_\pi = \infty$, meaning that making $\pi$ available in $x$ does not change the outcome of filtering. Regarding the $k$-SDP, this is the case where either (1) $\pi$ does not end in $s$, or (2) $x$ contains $k$ other $v$-$s$-paths that are shorter than $\pi$ or have the same weight as $\pi$ but are lexicographically ordered before $\pi$. While the notion of domination may seem overly complicated for the matter at hand, it sufficiently generalizes the proof for Lemma 14.27 to cover the $k$-SDP as well as the $k$-DSDP from Examples 14.28 and 14.29, respectively.

**Observation 14.26.** *Note that $r$ discards all paths not ending in $s$. Furthermore, for all paths $\pi$, we have*

$$r(x)_\pi = x_\pi \quad \text{or} \quad r(x)_\pi = \infty, \tag{14.55}$$

*i.e., $r(x) \in \mathcal{P}_{\min,+}(G)$ for all $x \in \mathcal{P}_{\min,+}(G)$, as $r$ either keeps a path with its original weight or discards it by setting the according entry to $\infty$. We also obtain that, for all $v \in V$,*

$$P_k(v, s, x) = P_k(v, s, r(x)) \tag{14.56}$$

*because $P_k(v, s, x)$ is invariant under discarding dominated paths from $x$.*

Following the standard approach — Lemma 13.8 — we define vectors $x, y \in \mathcal{P}_{\min,+}(G)$ to be equivalent if and only if their entries for $P_k(\cdot, s, x)$ do not differ:

$$\forall x, y \in \mathcal{P}_{\min,+}(G)\colon \quad x \sim y \quad :\Leftrightarrow \quad r(x) = r(y). \tag{14.57}$$

**Lemma 14.27.** *$\sim$ is a congruence relation on $\mathcal{P}_{\min,+}(G)$ with representative projection $r$.*

*Proof.* Clearly, $r$ is a projection. Below, we show in one step each that it fulfills Conditions (13.12) and (13.13) of Lemma 13.8. Throughout the proof, fix a graph $G = (V, E, \omega)$, let $x, x', y, y' \in \mathcal{P}_{\min,+}(G)$ be such that $x \sim x'$ and $y \sim y'$.

(1) We show below that $r(yx) = r(yr(x))$. Equation (13.12) then follows using that $r(yx) = r(yr(x)) = r(yr(x')) = r(yx')$. To see that $r(yx) = r(yr(x))$, we argue that for all $v$-$s$-paths $\pi$, we either have $(yx)_\pi = (yr(x))_\pi$, or both $\pi \notin P_k(v, s, yx)$ and $\pi \notin P_k(v, s, yr(x))$. In other words, the entries regarding $\pi$ are either equal in $yx$ and $yr(x)$, or $r$ discards $\pi$ from $yx$ as well as from $yr(x)$.

Consider a $v$-$s$-path $\pi$ with $(yx)_\pi \neq (yr(x))_\pi$. Observe that this implies $\omega(\pi) = (yx)_\pi < (yr(x))_\pi = \infty$ because the non-$\infty$ entries of $r(x)$ are a subset of those of $x$. Hence, $\pi$ is contained in $yx$. By definition of $\odot$, it holds that $(yx)_\pi = y_{\pi^1} + x_{\pi^2}$ for some partition $\pi = \pi^1 \circ \pi^2$, where $\pi^1$ and $\pi^2$ are, for some node $w$, $v$-$w$- and $w$-$s$-paths, respectively. It follows from Equation (14.55) and $x \in \mathcal{P}_{\min,+}(G)$ that

$$\omega(\pi^2) = x_{\pi^2} < r(x)_{\pi^2} = \infty. \tag{14.58}$$

Hence, $\pi^2 \notin P_k(w, s, x)$, i.e., $\pi^2$ is dominated in $x$. As $P_k(w, s, r(x)) = P_k(w, s, x)$ by Equation (14.56), the following $k$ $v$-$s$-paths are contained in $yr(x)$ and in $yx$ each:

$$\left\{\pi^1 \circ \bar{\pi}^2 \mid \bar{\pi}^2 \in P_k(w, s, r(x))\right\}. \tag{14.59}$$

We conclude that $\pi$ is dominated in — and thus discarded from — both $yx$ and $yr(x)$, as claimed.

(2) Consider a node $v \in V$ and a $v$-$s$-path $\pi$. As $(x \oplus y)_\pi = \min\{x_\pi, y_\pi\}$, we have $P(v, s, x \oplus y) = P(v, s, x) \cup P(v, s, y)$. In particular, it holds that $P_k(v, s, x \oplus y) \subseteq P_k(v, s, x) \cup P_k(v, s, y)$, because if $\pi \in P(v, s, x \oplus y)$ is dominated in $x$ and in $y$, it is dominated in $x \oplus y$ as well. Using Equation (14.56), we obtain that

$$P_k(v, s, x \oplus y) \subseteq P_k(v, s, x) \cup P_k(v, s, y) = P_k(v, s, r(x)) \cup P_k(v, s, r(y)). \tag{14.60}$$

As $r$ discards all paths not ending in $s$, and $x_\pi = r(x)_\pi$ for all $\pi \in P_k(v, s, r(x))$ and, analogously, $y_\pi = r(y)_\pi$ for all $\pi \in P_k(v, s, r(y))$, we conclude that $r(x \oplus y) = r(r(x) \oplus r(y))$. Hence, $r(x \oplus y) = r(r(x) \oplus r(y)) = r(r(x') \oplus r(y')) = r(x' \oplus y')$, i.e., $r$ fulfills Equation (13.13).

Since $x \sim x'$ and $y \sim y'$ are arbitrary, $r$ fulfills the preconditions of Lemma 13.8 and the claim follows. □

Observe that $r$ is defined such that it maintains the $k$ shortest $v$-$s$-paths *for all $v \in V$*, potentially storing $k|V|$ paths instead of just $k$. Intuitively, one could argue that $r^V x_v^{(h)}$ *only* needs to contain $k$ paths since they all start in $v$, which is what the algorithm should actually be doing. This objection is correct in that this is what actually happens when running the algorithm with initialization $x^{(0)}$: By Lemma 14.21, $x_v^{(h)}$ contains the $h$-hop shortest paths *starting in* $v$ and $r$ removes all that do not end in $s$ or are too long. On the other hand, the objection is misleading. In order for $r$ to behave correctly w.r.t. all

$x \in \mathcal{P}_{\min,+}(G)$ — including those less nicely structured than $x_v^{(h)}$ where all paths start at $v$ — we must define $r$ as it is, otherwise the proof of Lemma 14.27 fails for mixed starting-node inputs.

**Example 14.28** ($k$-Shortest Distance Problem)**.** *The k-SDP, compare Definition 14.22, is solved by an MBF-like algorithm $\mathcal{A}$ with $\mathcal{S} = \mathcal{M} = \mathcal{P}_{\min,+}(G)$, the representative projection and congruence relation defined in Equations* (14.54) *and* (14.57)*, the choices of $A$ and $x^{(0)}$ from Equations* (14.45) *and* (14.46)*, and $h = \mathrm{SPD}(G)$ iterations.*

By Lemma 14.21 and due to $h = \mathrm{SPD}(G)$, $x_v^{(h)}$ contains all $v$-$s$-paths with their correct weights. Hence, $\mathcal{A}^h(G)_v = (r^V x^{(h)})_v$ contains the subset of those paths the $k$-SDP asks for. We remark that solving a generalization of the $k$-SDP looking for the $k$ shortest $h$-hop distances is straightforward using $h$ iterations. Furthermore, note that our approach reveals the actual paths along with their weights.

**Example 14.29** ($k$-Distinct-Shortest Distance Problem)**.** *The k-DSDP from Definition 14.22 is solved by an MBF-like algorithm analogous to that for the k-SDP from Example 14.28.*

In order for this to work, the definition of $P_k(v,w,x)$ in Equation (14.53) needs to be adjusted. For each of the $k$ smallest weights in $x$, the modified $\bar{P}_k(v,w,x)$ contains only one representative: the path contained in $x$ of that weight that is first w.r.t. lexicographical order on $P$. This results in

$$\bar{P}'_k(v,w,x) := \{\pi \mid x_\pi \text{ is among the } k \text{ smallest of } \{x_{\pi'} \mid \pi' \in P(v,w,x)\}\} \text{ and} \tag{14.61}$$
$$\bar{P}_k(v,w,x) := \{\pi \mid \pi \text{ is lexicographically first of } \{\pi' \in \bar{P}'_k(v,w,x) \mid x_{\pi'} = x_\pi\}\}. \tag{14.62}$$

Observe that this changes the meaning of $\pi$ being dominated in $x$. With that in mind, Observation 14.26 and the proof of Lemma 14.27 work without modification when replacing Equation (14.53) with Equations (14.61)–(14.62).

## 14.4 MBF-like Algorithms over the Boolean Semiring

A well-known semiring is the Boolean semiring $\mathcal{B} := (\mathbb{B}, \vee, \wedge)$ and by Lemma 2.4, $\mathcal{B}^V$ is a zero-preserving semimodule over $\mathcal{B}$. It can be used to check for connectivity in a graph[1] using the adjacency matrix

$$(a_{vw}) := \begin{cases} 1 & \text{if } v = w, \\ 1 & \text{if } \{v,w\} \in E, \text{ and} \\ 0 & \text{otherwise} \end{cases} \tag{14.63}$$

together with initial values

$$x_{vw}^{(0)} := \begin{cases} 1 & \text{if } v = w \text{ and} \\ 0 & \text{otherwise} \end{cases} \tag{14.64}$$

indicating that each node $v \in V$ is connected to itself. Induction over $h$ reveals that

$$\left(A^h x^{(0)}\right)_{vw} = 1 \quad \Leftrightarrow \quad \mathrm{P}^h(v,w,G) \neq \emptyset. \tag{14.65}$$

[1]In this section, we drop the assumption that graphs are connected.

**Example 14.30** (Connectivity)**.** *Given a graph, we want to check which pairs of nodes are connected by paths of at most $h$ hops. This is solved by an MBF-like algorithm using $\mathcal{S} = \mathcal{B}$, $\mathcal{M} = \mathcal{B}^V$, $r = \mathrm{id}$, and $x^{(0)}$ from Equation* (14.64)*.*

Example 14.30 directly generalizes to single-source and multi-source connectivity variants using a filter function that discards non-sources.

# CHAPTER 15

# An Oracle for MBF-like Queries

Our goal is to solve distance problems in a graph $G$ using polylog $n$ depth and as little work as possible. In order to keep the number of iterations low, we augment $G$ with Cohen's $(d, \hat{\varepsilon})$-hop set [34]: a small number of additional (weighted) edges for $G$, such that for all $v, w \in V$, $\operatorname{dist}^d(v, w, G) \leq (1 + \hat{\varepsilon}) \operatorname{dist}(v, w, G)$, where $d, \hat{\varepsilon}^{-1} \in \operatorname{polylog} n$. Cohen's algorithm is sufficiently efficient in terms of depth, work, and number of additional edges. Our presentation, however, is independent from the hop-set algorithm: We only require that $\operatorname{dist}^d(\cdot, \cdot, G)$ is a $(1 + \hat{\varepsilon})$-approximation of $\operatorname{dist}(\cdot, \cdot, G)$, i.e., that $d$ iterations of MBF-like algorithms obtain good distance approximations, which is a cornerstone for algorithms of polylog $n$ depth. As detailed in Observation 12.1, however, $\operatorname{dist}^d(\cdot, \cdot, G)$ does not obey the triangle inequality and $\operatorname{SPD}(G)$ remains unaffected by additional hop-set edges. Unfortunately, the triangle inequality is vital to the problems that we discuss: approximate metrics and FRT embeddings in Chapters 16 and 17, respectively.

Hence, we reduce the SPD, accepting a slight increase in stretch. After augmenting $G$ with the hop set, we embed it into a complete graph $H$ on the same node set such that $\operatorname{SPD}(H) \in \operatorname{O}(\log^2 n)$ and $\operatorname{dist}(v, w, G) \leq \operatorname{dist}(v, w, H) \in (1 + \operatorname{o}(1)) \operatorname{dist}(v, w, G)$. Where hop sets preserve distances *exactly* and ensure the existence of *approximately* shortest paths with few hops, $H$ preserves distances *approximately* but guarantees that we obtain *exact* shortest paths with few hops. This is discussed in Section 15.1.

This approach keeps algorithms of polylog $n$ depth in reach, resolves the issue with the triangle inequality, and still guarantees $(1 + \operatorname{o}(1))$-approximate distances w.r.t. $G$. Unfortunately, however, $H$ is a complete graph, so directly working on it or even storing it in memory imposes $\Omega(n^2)$ work. Hence, we improve our approach to allow low-work algorithms while still maintaining the requirement of polylog $n$ depth. In particular, we want to sample an FRT embedding using near-linear work in $m$, see Chapter 17. Our solution is an oracle for MBF-like queries over $\mathcal{S}_{\min,+}$; we develop it in Sections 15.2–15.3. Given an MBF-like algorithm $\mathcal{A}$ over $\mathcal{S}_{\min,+}$, a graph $G$, and a number of iterations $h$, the oracle answers with $\mathcal{A}^h(H)$. Put differently, the oracle can simulate $\mathcal{A}$ in $H$ without explicitly constructing $H$, instead, it internally works on $G$. The oracle heavily exploits the properties of MBF-like algorithms from Chapter 13 and simulates each iteration of $\mathcal{A}$ in $H$ using polylogarithmically many iterations of $\mathcal{A}$ in $G$. As $h = \operatorname{SPD}(H) \in \operatorname{O}(\log^2 n)$ iterations suffice, $\mathcal{A}^h(H)$ can be determined with only a *polylogarithmic overhead w.r.t. a single iteration in $G$.* This keeps a work of $\tilde{\operatorname{O}}(m)$ in reach while maintaining polylog $n$ depth, obeying the triangle inequality, and $(1 + \operatorname{o}(1))$-approximating distances.

## 15.1 The Simulated Graph $H$

After augmenting $G$ with the hop set, we embed it into a complete graph $H$ on the same node set so that $\operatorname{SPD}(H) \in \operatorname{O}(\log^2 n)$, keeping the stretch limited. Since our construction requires to first add a hop set to $G$, assume for the sake of presentation that $G$ already contains a $(d, \hat{\varepsilon})$-hop set for fixed $\hat{\varepsilon} \in \mathbb{R}_{>0}$ and $d \in \mathbb{N}$ throughout this chapter.

We begin the construction of $H$ by sampling levels for the vertices $V$: Every vertex starts at level 0. In step $\lambda \geq 1$, each vertex in level $\lambda - 1$ is raised to level $\lambda$ with probability $\frac{1}{2}$. We continue until the first step $\Lambda + 1$ where no node is sampled. $\lambda(v)$ refers to the level of $v \in V$ and we define the level of an edge $e \in E$ as $\lambda(e) := \min\{\lambda(v) \mid v \in e\}$, the minimal level of its incident vertices. In the following, we abbreviate $\lambda(v, w) := \lambda(\{v, w\})$.

**Lemma 15.1.** *W.h.p., $\Lambda \in \mathrm{O}(\log n)$.*

*Proof.* For a constant $c \in \mathbb{R}_{\geq 1}$, $v \in V$ has $\lambda(v) < c \log n \in \mathrm{O}(\log n)$ with a probability of at least $1 - (\frac{1}{2})^{c \log n} = 1 - n^{-c}$, i.e., w.h.p. By Lemma 2.7, this w.h.p. holds for all nodes and the claim follows. □

The idea is to use the levels in the following way. We devise a complete graph $H$ on $V$. An edge of $H$ of level $\lambda$ is weighted with the $d$-hop distance between its endpoints in $G$ — a $(1+\hat{\varepsilon})$-approximation of their exact distance because $G$ contains a $(d, \hat{\varepsilon})$-hop set by assumption — multiplied with a penalty of $(1 + \hat{\varepsilon})^{\Lambda - \lambda}$. This way, high-level edges are "more attractive" for shortest paths, because they receive exponentially smaller penalties.

**Definition 15.2** (Simulated graph $H$)**.** *Let $G = (V, E, \omega)$ be a graph that contains a $(d, \hat{\varepsilon})$-hop set with levels sampled as above. We define the complete graph $H$ as*

$$H := \left(V, \binom{V}{2}, \omega_\Lambda\right) \tag{15.1}$$

$$\omega_\Lambda(\{v, w\}) := (1 + \hat{\varepsilon})^{\Lambda - \lambda(v,w)} \operatorname{dist}^d(v, w, G). \tag{15.2}$$

We formalize the notion of high-level edges being "more attractive" than low-level paths. In $H$, any min-hop shortest $v$-$w$-path is exclusively comprised of edges of level $\lambda(v, w)$ or higher; no min-hop shortest path's level locally decreases. Therefore, all min-hop shortest paths can be split into two subpaths, the first of nondecreasing and the second of nonincreasing level.

**Lemma 15.3.** *Consider $v, w \in V$, $\lambda = \lambda(v, w)$, and $p \in \mathrm{MHSP}(v, w, H)$. Then all edges of $p$ have level at least $\lambda$.*

*Proof.* The case $\lambda = 0$ is trivial. Consider $1 \leq \lambda \leq \Lambda$ and, for the sake of contradiction, let $q$ be a non-trivial maximal subpath of $p$ containing only edges of level strictly less than $\lambda$. Observe that $q \in \mathrm{MHSP}(v', w', H)$ for some $v', w' \in V$ with $\lambda(v'), \lambda(w') \geq \lambda$. We have

$$\omega_\Lambda(q) \geq (1 + \hat{\varepsilon})^{\Lambda - (\lambda - 1)} \operatorname{dist}(v', w', G) \tag{15.3}$$

because $q$ only uses edges of level at most $\lambda - 1$, which all receive a multiplicative penalty of at least $(1 + \hat{\varepsilon})^{\Lambda - (\lambda - 1)}$ in $\omega_\Lambda$. However, the edge $e = \{v', w'\}$ has level $\lambda(v', w') \geq \lambda$ and weight

$$\omega_\Lambda(e) \leq (1 + \hat{\varepsilon})^{\Lambda - \lambda} \operatorname{dist}^d(v', w', G) \leq (1 + \hat{\varepsilon})^{\Lambda - (\lambda - 1)} \operatorname{dist}(v', w', G) \leq \omega_\Lambda(q) \tag{15.4}$$

because $G$ contains a $(d, \hat{\varepsilon})$-hop set. As $q$ is a min-hop shortest path, we have $q = \{e\}$, contradicting the assumption that $q$ only contains edges of level $\lambda - 1$ or less. □

Having established that edge levels in min-hop shortest paths are first nondecreasing and then nonincreasing, the next step is to limit the number of hops spent on each level.

**Lemma 15.4.** *Consider vertices $v$ and $w$ of $H$ with $\lambda(v), \lambda(w) \geq \lambda$. Then w.h.p., one of the following statements holds:*

$$\mathrm{hop}(v, w, H) \in \mathrm{O}(\log n) \text{ or} \tag{15.5}$$

$$\forall p \in \mathrm{MHSP}(v, w, H)\ \exists e \in p\colon\ \lambda(e) \geq \lambda + 1. \tag{15.6}$$

*Proof.* Condition on the event $\mathcal{E}_{V_\lambda}$ that $V_\lambda \subseteq V$, with $v, w \in V_\lambda$, is the set of nodes with level $\lambda$ or higher, with level $\lambda + 1$ not yet sampled. Let

$$H_\lambda := \left(V_\lambda, \binom{V_\lambda}{2}, \omega_\lambda\right) \text{ with} \tag{15.7}$$

$$\omega_\lambda(\{v, w\}) := (1 + \hat{\varepsilon})^{\Lambda - \lambda} \operatorname{dist}^d(v, w, G) \tag{15.8}$$

denote the subgraph of $H$ spanned by $V_\lambda$ and capped at level $\lambda$.

Consider $p \in \mathrm{MHSP}(v, w, H_\lambda)$. Observe that $\mathbb{P}[\lambda(u) \geq \lambda + 1 \mid \mathcal{E}_{V_\lambda}] = \frac{1}{2}$ independently for all $u \in V_\lambda$, and hence $\mathbb{P}[\lambda(e) \geq \lambda + 1 \mid \mathcal{E}_{V_\lambda}] = \frac{1}{4}$ for all $e \in p$. The latter probability holds independently for every other edge of $p$. If $|p| \geq 2c \log_{4/3} n + 1$ for some choice of $c \in \mathbb{R}_{\geq 1}$, the probability that $p$ contains no edge of level $\lambda + 1$ or higher is bounded from above by $(\frac{3}{4})^{\lfloor |p|/2 \rfloor} \leq (\frac{3}{4})^{(|p|-1)/2} = (\frac{3}{4})^{c \log_{4/3} n} = n^{-c}$, so $p$ contains such an edge w.h.p.

Fix an arbitrary $p \in \mathrm{MHSP}(v, w, H_\lambda)$. Let $\mathcal{E}_p$ denote the event that $p$ contains an edge of level at least $\lambda + 1$ or fulfills $|p| \in \mathrm{O}(\log n)$; as argued above, $\mathcal{E}_p$ occurs w.h.p. Note that we cannot directly apply the union bound and Lemma 2.7 to deduce a similar statement for all $q \in \mathrm{MHSP}(v, w, H_\lambda)$ because $|\mathrm{MHSP}(v, w, H_\lambda)|$ is not polynomially bounded. Instead, we argue that if $\mathcal{E}_p$ holds, it follows that all $q \in \mathrm{MHSP}(v, w, H)$ must behave as claimed.

To show that all $q \in \mathrm{MHSP}(v, w, H)$ fulfill (15.5) or (15.6) under the assumption that $\mathcal{E}_p$ holds for the choice of $p$ fixed above, recall that $q$ only uses edges of level $\lambda$ or higher by Lemma 15.3. Furthermore, observe that $\omega_\Lambda(q) \leq \omega_\Lambda(p)$ because $q$ is a shortest path in $H$, that $\omega_\lambda(p) \leq \omega_\lambda(q)$ because $p$ is a shortest path in $H_\lambda$, and that $\omega_\Lambda(p) \leq \omega_\lambda(p)$ for all $p \in \mathrm{P}(v, w, H_\lambda)$ by construction of $H$ and $H_\lambda$. If $q$ contains an edge of level $\lambda + 1$ or higher, (15.6) holds for $q$. Otherwise, $q$ uses only edges that are exactly of level $\lambda$. We conclude $\omega_\lambda(q) = \omega_\Lambda(q)$ and distinguish two cases:

**Case 1** If $|p| \in \mathrm{O}(\log n)$, we have

$$\omega_\Lambda(p) \leq \omega_\lambda(p) \leq \omega_\lambda(q) = \omega_\Lambda(q), \tag{15.9}$$

and $\omega_\Lambda(q) = \omega_\Lambda(p)$ and $|q| \leq |p|$ both follow from $q \in \mathrm{MHSP}(v, w, H)$. It follows from the case premise that $|q| \in \mathrm{O}(\log n)$, i.e., that (15.5) holds.

**Case 2** If $p$ contains an edge of level $\lambda + 1$ or higher, we obtain $\omega_\Lambda(p) < \omega_\lambda(p)$. This implies

$$\omega_\Lambda(p) < \omega_\lambda(p) \leq \omega_\lambda(q) = \omega_\Lambda(q), \tag{15.10}$$

which contradicts $q \in \mathrm{MHSP}(v, w, H)$.

So far, we condition on $\mathcal{E}_{V_\lambda}$. In order to remove this restriction, fix $v, w \in V$ and let $\mathcal{E}_{vw}$ denote the event that (15.5) or (15.6) holds for $v$ and $w$. The above case distinction shows that $\mathbb{P}[\mathcal{E}_{vw} \mid \mathcal{E}_{V_\lambda}] \geq 1 - n^{-c}$ for an arbitrary $c \in \mathbb{R}_{\geq 1}$. We conclude that

$$\begin{aligned}
\mathbb{P}[\mathcal{E}_{vw} \mid \lambda(v,w) \geq \lambda] &= \sum_{V_\lambda \subseteq V} \mathbb{P}[\mathcal{E}_{V_\lambda} \mid \lambda(v,w) \geq \lambda]\, \mathbb{P}[\mathcal{E}_{vw} \mid \mathcal{E}_{V_\lambda}] && (15.11)\\
&= \sum_{\{v,w\} \subseteq V_\lambda \subseteq V} \mathbb{P}[\mathcal{E}_{V_\lambda} \mid \lambda(v,w) \geq \lambda]\, \mathbb{P}[\mathcal{E}_{vw} \mid \mathcal{E}_{V_\lambda}] && (15.12)\\
&\geq \sum_{\{v,w\} \subseteq V_\lambda \subseteq V} \mathbb{P}[\mathcal{E}_{V_\lambda} \mid \lambda(v,w) \geq \lambda](1 - n^{-c}) && (15.13)\\
&= (1 - n^{-c}) \sum_{\{v,w\} \subseteq V_\lambda \subseteq V} \mathbb{P}[\mathcal{E}_{V_\lambda} \mid \lambda(v,w) \geq \lambda] && (15.14)\\
&= 1 - n^{-c}, && (15.15)
\end{aligned}$$

which is the statement of the lemma. □

We argue that any min-hop shortest path in $H$ traverses every level at most twice in Lemma 15.3, Lemma 15.4 states that each such traversal, w.h.p., only has a logarithmic number of hops, and Lemma 15.1 asserts that, w.h.p., there are only logarithmically many levels. Together, this means that min-hop shortest paths in $H$ have $\mathrm{O}(\log^2 n)$ hops w.h.p. Additionally, our construction limits the stretch of shortest paths in $H$ as compared to $G$ by $(1+\hat{\varepsilon})^{\Lambda+1}$, i.e., by $(1+\hat{\varepsilon})^{\mathrm{O}(\log n)}$ w.h.p.

**Theorem 15.5.** *W.h.p.,* $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ *and, for all* $v, w \in V$,

$$\operatorname{dist}(v,w,G) \leq \operatorname{dist}(v,w,H) \leq (1+\hat{\varepsilon})^{\mathrm{O}(\log n)} \operatorname{dist}(v,w,G). \tag{15.16}$$

*Proof.* Fix a level $\lambda$. Any fixed pair of vertices of level $\lambda$ or higher fulfills, w.h.p., (15.5) or (15.6) by Lemma 15.4. Since there are at most $\binom{n}{2}$ such pairs, w.h.p., all of them fulfill (15.5) or (15.6) by Lemma 2.7.

Let $\mathcal{E}_{\log}$ denote the event that there is no higher level than $\Lambda \in \mathrm{O}(\log n)$, which holds w.h.p. by Lemma 15.1. Furthermore, let $\mathcal{E}_\lambda$ denote the event that all pairs of vertices of level $\lambda$ or higher fulfill (15.5) or (15.6), which holds w.h.p. as argued above. Then $\mathcal{E} := \mathcal{E}_{\log} \cap \mathcal{E}_0 \cap \dots \cap \mathcal{E}_\Lambda$ holds w.h.p. by Lemma 2.7.

Condition on $\mathcal{E}$. In particular, no min-hop shortest path whose edges all have the same level has more than $\mathrm{O}(\log n)$ hops. Consider some min-hop shortest path $p$ in $H$. By Lemma 15.3, $p$ has two parts: The edge level is nondecreasing in the first and nonincreasing in the second part. Hence, $p$ can be split into at most $2\Lambda - 1$ maximal segments, in each of which all edges have the same level. As this holds for all min-hop shortest paths, we conclude that $\mathrm{SPD}(H) \in \mathrm{O}(\Lambda \log n) \subseteq \mathrm{O}(\log^2 n)$, as claimed.

As for Inequality (15.16), recall that $H$ is constructed from $G = (V, E, \omega)$, and that $G$ contains a $(d, \hat{\varepsilon})$-hop set. For all $v, w \in V$, we have

$$\operatorname{dist}(v,w,H) \leq \omega_\Lambda(v,w) \leq (1+\hat{\varepsilon})^{\Lambda} \operatorname{dist}^d(v,w,G) \leq (1+\hat{\varepsilon})^{\Lambda+1} \operatorname{dist}(v,w,G) \tag{15.17}$$

and $\operatorname{dist}(v,w,H) \geq \operatorname{dist}(v,w,G)$ by construction of $H$. Recalling that $\Lambda \in \mathrm{O}(\log n)$ and that we only condition on $\mathcal{E}$, which occurs w.h.p., completes the proof. □

We use Cohen's construction to obtain a $(d,\hat{\varepsilon})$-hop set with $\hat{\varepsilon} \in 1/\operatorname{polylog} n$, where the exponent of the $\operatorname{polylog} n$ term is under our control [34]. A sufficiently large exponent yields

$$(1+\hat{\varepsilon})^{\mathrm{O}(\log n)} \overset{1+x \leq e^x}{\subseteq} e^{\hat{\varepsilon}\,\mathrm{O}(\log n)} \subseteq e^{1/\operatorname{polylog} n} \subseteq 1 + 1/\operatorname{polylog} n, \tag{15.18}$$

where the last step follows from observing that $e^x \leq 1 + e^u x$ for all $0 \leq x \leq u$. This upper-bounds (15.16) by

$$\operatorname{dist}(v,w,G) \leq \operatorname{dist}(v,w,H) \in (1 + 1/\operatorname{polylog} n)\operatorname{dist}(v,w,G). \tag{15.19}$$

To sum up: Given a weighted graph $G$, we augment $G$ with a $(d, 1/\operatorname{polylog} n)$-hop set. After that, the $d$-hop distances in $G$ readily $(1 + \mathrm{o}(1))$-approximate the actual distances in $G$, but these approximations may violate the triangle inequality. We fix this by embedding into $H$, using geometrically sampled node levels and an exponential penalty on the edge weights with decreasing levels. The result is that where $G$ preserves distances exactly and ensures the existence of approximate shortest paths with few hops, $H$ preserves distances $(1 + \mathrm{o}(1))$-approximately but guarantees that we obtain exact shortest paths with few hops.

## 15.2 Decomposing $H$

The situation is that $H$ has a polylogarithmic SPD — more precisely, $\operatorname{SPD}(H) \in \mathrm{O}(\log^2 n)$ w.h.p. — while at the same time $(1 + \mathrm{o}(1))$-approximating the distances in $G$. This paves the way for algorithms of $\operatorname{polylog} n$ depth. In order to keep the work under control, however, recall that explicitly writing $H$ into memory to directly work on it imposes $\Omega(n^2)$ work which rules out subquadratic-work algorithms. In particular, this prevents us from sampling an FRT embedding using near-linear work in $m$, see Chapter 17.

The idea to fix this is to never explicitly write $H$ — $H$ is uniquely determined by $G$ and the node levels — and to simulate iterations of an MBF-like algorithm $\mathcal{A}$ in $H$, using $d$ iterations of $\mathcal{A}$ in $G$ instead. As $A_G$, the adjacency matrix of $G$, only has $\mathrm{O}(m)$ non-$\infty$ entries and $hd \in \operatorname{polylog} n$, this maintains our requirement of algorithms with $\operatorname{polylog} n$ depth that still may have a work bounded by $\tilde{\mathrm{O}}(m)$. To this end, we require a representation of $A_H$, the adjacency matrix of $H$, in terms of $A_G$.

For the remainder of this chapter, we keep the notation of $A_G$ and $A_H$ as adjacency matrices of $G$ and $H$, and fix the semiring to be $\mathcal{S}_{\min,+}$. In order to formalize the decomposition of $A_H$ in terms of $A_G$ motivated above, recall that by Definition 15.2 we have

$$(A_H)_{vw} = \omega_\Lambda(v,w) = (1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)} \operatorname{dist}^d(v,w,G) = (1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)} (A_G^d)_{vw}. \tag{15.20}$$

For a level $\lambda \in \{0,\dots,\Lambda\}$, denote by $P_\lambda \colon \mathcal{M}^V \to \mathcal{M}^V$ the projection to coordinates of level $\lambda$ or higher, i.e., to $V_\lambda := \{v \in V \mid \lambda(v) \geq \lambda\}$:

$$P_\lambda(x)_v := \begin{cases} x_v & \text{if } \lambda(v) \geq \lambda \text{ and} \\ \bot & \text{otherwise.} \end{cases} \tag{15.21}$$

Observe that $P_\lambda$ is an SLF with $(P_\lambda)_{vw} = 0$ if $v = w \in V_\lambda$ and $(P_\lambda)_{vw} = \infty$ otherwise. Furthermore, let $A_\lambda$ be the adjacency matrix $A_G$ stretched by a factor of $(1+\hat{\varepsilon})^{\Lambda-\lambda}$, corresponding to the stretch imposed on edges of level $\lambda$ in $H$:

$$(A_\lambda)_{vw} := (1+\hat{\varepsilon})^{\Lambda-\lambda}(A_G)_{vw}. \tag{15.22}$$

This gives us the tools to decompose $A_H$ as motivated above.

**Lemma 15.6.** *With the above definitions of $A_\lambda$ and $P_\lambda$, we have*

$$A_H = \bigoplus_{\lambda=0}^{\Lambda} P_\lambda A_\lambda^d P_\lambda. \tag{15.23}$$

*Proof.* Since $(A_G^d)_{vw} = \operatorname{dist}^d(v,w,G)$, we have $(A_\lambda^d)_{vw} = (1+\hat{\varepsilon})^{\Lambda-\lambda}\operatorname{dist}^d(v,w,G)$. Therefore, we get

$$\begin{align}
\left(A_\lambda^d P_\lambda\right)_{vw} &= \min_{u\in V}\left\{(A_\lambda^d)_{vu} + (P_\lambda)_{uw}\right\} \tag{15.24}\\
&= \begin{cases}(1+\hat{\varepsilon})^{\Lambda-\lambda}\operatorname{dist}^d(v,w,G) & \text{if } w\in V_\lambda \text{ and}\\ \infty & \text{otherwise,}\end{cases} \tag{15.25}
\end{align}$$

and hence

$$\begin{align}
\left(P_\lambda A_\lambda^d P_\lambda\right)_{vw} &= \min_{u\in V}\left\{(P_\lambda)_{vu} + (A_\lambda^d P_\lambda)_{uw}\right\} \tag{15.26}\\
&= \begin{cases}(1+\hat{\varepsilon})^{\Lambda-\lambda}\operatorname{dist}^d(v,w,G) & \text{if } v,w\in V_\lambda \text{ and}\\ \infty & \text{otherwise.}\end{cases} \tag{15.27}
\end{align}$$

We conclude that

$$\begin{align}
\left(\bigoplus_{\lambda=0}^{\Lambda} P_\lambda A_\lambda^d P_\lambda\right)_{vw} &= \min_{\lambda=0}^{\lambda(v,w)}\left\{(1+\hat{\varepsilon})^{\Lambda-\lambda}\operatorname{dist}^d(v,w,G)\right\} \tag{15.28}\\
&= (1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)}\operatorname{dist}^d(v,w,G) \tag{15.29}\\
&= \omega_\Lambda(v,w) \tag{15.30}\\
&= (A_H)_{vw}. \qquad \square
\end{align}$$

As the oracle computes $\mathcal{A}^h(H)$, i.e, $r^V A_H^h x^{(0)}$, we use our decomposition of $A_H$ to analyze $A_H^h$ in that regard. With an efficient implementation in mind, we take the freedom to apply filters intermediately, which is feasible due to $r^V \sim \operatorname{id}$ by Corollary 13.20. It is important to keep in mind, however, that we do not compute any of these matrix multiplications explicitly. Instead, we determine $(r^V A_\lambda)^d x$ using $\mathcal{A}^d(G)$, i.e., $d$ iterations of $\mathcal{A}$ in $G$. Filtering, projection, and aggregation are computed directly. For all $h \in \mathbb{N}_0$ we have

$$A_H^h \overset{(15.23)}{=} \left(\bigoplus_{\lambda=0}^{\Lambda} P_\lambda A_\lambda^d P_\lambda\right)^h \overset{(13.39)}{\sim} \left(r^V\left(\bigoplus_{\lambda=0}^{\Lambda} P_\lambda (r^V A_\lambda)^d P_\lambda\right)\right)^h r^V \tag{15.31}$$

```
input: A graph $G = (V, E, \omega)$ containing a $(d, \hat{\varepsilon})$-hop set and an MBF-like algorithm $\mathcal{A}$
    over $\mathcal{S}_{\min,+}$ with initialization $x^{(0)}$ and $h \in \mathbb{N}_0$ iterations
output: $\mathcal{A}^h(H)$
 1: $x^{(0)} \leftarrow r^V x^{(0)}$
 2: for $i \leftarrow 1, \dots, h$ do                                ▷ $h$ iterations in $H$
 3:     for each $\lambda \in \{0, \dots, \Lambda\}$ in parallel do
 4:         $y_\lambda \leftarrow P_\lambda x^{(i-1)}$                         ▷ discard low-level nodes
 5:         for $f \leftarrow 1, \dots, d$ do    ▷ $d$ iterations of $\mathcal{A}$ in $G$ with stretched edge weights
 6:             $y_\lambda \leftarrow r^V A_\lambda y_\lambda$
 7:         end for
 8:         $y_\lambda \leftarrow P_\lambda y_\lambda$                             ▷ discard low-level nodes
 9:     end for
10:     $x^{(i)} \leftarrow r^V \bigoplus_{\lambda=0}^{\Lambda} y_\lambda$
11: end for
12: return $x^{(h)}$
```

**Algorithm 15.1:** An Oracle for MBF-like Algorithms over $\mathcal{S}_{\min,+}$.

and hence

$$\mathcal{A}^h(H) \overset{(13.18)}{=} r^V A_H^h x^{(0)} \overset{(13.11),(15.31)}{=} \left( r^V \left( \bigoplus_{\lambda=0}^{\Lambda} P_\lambda (r^V A_\lambda)^d P_\lambda \right) \right)^h r^V x^{(0)}. \tag{15.32}$$

Observe that we can choose $h = \mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ w.h.p. by Theorem 15.5 and recall that w.h.p. $\Lambda \in \mathrm{O}(\log n)$ by Lemma 15.1 and that $d \in \operatorname{polylog} n$ for an appropriate hop set [34]. Overall, we obtain $hd\Lambda \in \operatorname{polylog} n$ which allows us to determine $\mathcal{A}(H)$ with polylogarithmic depth and $\tilde{\mathrm{O}}(m)$ work, provided that the individual steps of the underlying MBF-like algorithm can be implemented at this complexity.

## 15.3 Implementation

The oracle determines iterations of $\mathcal{A}$ in $H$ using iterations in $G$ while only introducing a polylogarithmic overhead w.r.t. iterations in $G$. With the decomposition of $H$ from Lemma 15.6 at hand, we propose Algorithm 15.1 as an implementation of the oracle. It works as follows. Given a state vector $x^{(i-1)} \in \mathcal{M}^V$ representing the result of the $(i-1)$-th iteration, simulate the $i$-th iteration of $\mathcal{A}$ in $H$ for edges of level $\lambda$, i.e., determine $y_\lambda := P_\lambda (r^V A_\lambda)^d P_\lambda x^{(i-1)}$ by (1) discarding entries at nodes of a level smaller than $\lambda$ (line 4), (2) running $d$ iterations of $\mathcal{A}$ with distances stretched by $(1 + \hat{\varepsilon})^{\Lambda - \lambda}$ in $G$, applying the filter after each iteration (lines 5–7), and (3) again discarding entries at nodes with levels smaller than $\lambda$ (line 8). After running this procedure in parallel for all $0 \le \lambda \le \Lambda$, aggregate the results of all levels and apply the node-wise filter, i.e., determine $x^{(i)} = r^V (\bigoplus_{\lambda=0}^{\Lambda} y_\lambda)$ (line 10).

Consider nodes $v, w \in V$. Since $(A_\lambda^d)_{vw} = (1 + \hat{\varepsilon})^{\Lambda - \lambda} \operatorname{dist}^d(v, w, G)$, the above approach ensures that we propagate information between nodes $v$ and $w$ with the correct edge weight in the parallel run for $\lambda = \lambda(v, w)$. Furthermore, any exchange between $v$ and $w$ in any run for $\lambda > \lambda(v, w)$ is discarded by $P_\lambda$. In a run for $\lambda < \lambda(v, w)$, however,

$v$ and $w$ exchange information. But observe that this information is propagated over too long a distance because edge weights are scaled by $(1+\hat{\varepsilon})^{\Lambda-\lambda} > (1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)}$. Hence, the "outdated" information "loses" against the copy that was propagated over the shorter distance during aggregation. Formally, it is discarded due to the left-distributive law of semimodules over $\mathcal{S}_{\min,+}$:

$$\left((1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)} \odot x\right) \oplus \left((1+\hat{\varepsilon})^{\Lambda-\lambda} \odot x\right) \tag{15.33}$$

$$\overset{(2.9)}{=} \left((1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)} \oplus (1+\hat{\varepsilon})^{\Lambda-\lambda}\right) \odot x \tag{15.34}$$

$$= \min\left\{(1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)}, (1+\hat{\varepsilon})^{\Lambda-\lambda}\right\} \odot x \tag{15.35}$$

$$= (1+\hat{\varepsilon})^{\Lambda-\lambda(v,w)} \odot x \tag{15.36}$$

for all semimodules $\mathcal{M}$ over $\mathcal{S}_{\min,+}$ and all $x \in \mathcal{M}$. Therefore, aggregating the results of all levels (using $\oplus$) in line 10 and applying $r^V$ completes the simulation of an iteration of $\mathcal{A}$ in $H$.

The efficiency of Algorithm 15.1 crucially depends on directly implementing iterations of $\mathcal{A}$ in $G$ instead of explicitly multiplying matrices (line 6) as well as on the semimodule $\mathcal{M}$ and the filter $r$ used by $\mathcal{A}$. In order to keep Theorem 15.7 general, we do not impose restrictions on $\mathcal{M}$ or $r$. Hence, the work and depth of the basic operations — determining $r^V x^{(0)}$ from $x^{(0)}$ (line 1), $r^V A_\lambda y$ from an intermediate state $y$ (line 6), and $\bigoplus_{\lambda=0}^{\Lambda} y_\lambda$ from the individual $y_\lambda$ (line 10) — are parameters of the theorem. Observe that we could use single-purpose work and depth parameters for determining $r^V x^{(0)}$ from $x^{(0)}$ instead of reusing $W$ and $D$. This, however, usually is a trivial operation and we do not wish to obstruct the presentation with further parameters.

Together, we can run any MBF-like algorithm $\mathcal{A}$ over $\mathcal{S}_{\min,+}$ in $H$ with only a polylogarithmic overhead w.r.t. just *a single iteration of $\mathcal{A}$ in $G$.*

**Theorem 15.7** (Oracle)**.** *Consider an MBF-like algorithm $\mathcal{A}$ using a semimodule $\mathcal{M}$ over the semiring $\mathcal{S}_{\min,+}$, the representative projection $r\colon \mathcal{M} \to \mathcal{M}$, and the initialization $x^{(0)} \in \mathcal{M}^V$. If we can, for all $1 \le f \le d$, $1 \le i \le h$, and $0 \le \lambda \le \Lambda$, where we may assume $\Lambda \in \mathrm{O}(\log n)$,*

*(1) compute $r^V x^{(0)}$ from $x^{(0)}$ with depth $D$ and work $W$ (line 1),*

*(2) determine $r^V A_\lambda y$ from any intermediate state vector $y = (r^V A_\lambda)^{f-1} P_\lambda x^{(i-1)}$ — corresponding to the $f$-th iteration w.r.t. $A_\lambda$ starting at state $x^{(i-1)}$ — with depth $D$ and work $W$ (line 6), and*

*(3) compute $r^V(\bigoplus_{\lambda=0}^{\Lambda} y_\lambda)$ from the individual $y_\lambda = (P_\lambda r^V A_\lambda^d P_\lambda) x^{(i-1)}$, $0 \le \lambda \le \Lambda$ — reflecting aggregation over all levels to complete a simulated iteration in $H$ — using depth $D_\oplus$ and work $W_\oplus$ (line 10),*

*then we can w.h.p.*

*(1) determine $\mathcal{A}^h(H)$ using $\mathrm{O}((dW\log n + W_\oplus)h) \subseteq \tilde{\mathrm{O}}((dW + W_\oplus)h)$ work and a depth of $\mathrm{O}((dD + D_\oplus)h)$ and thus*

*(2) calculate $\mathcal{A}(H)$ using $\mathrm{O}((dW\log n + W_\oplus)\log^2 n) \subseteq \tilde{\mathrm{O}}(dW + W_\oplus)$ work and a depth of $\mathrm{O}((dD + D_\oplus)\log^2 n) \subseteq \tilde{\mathrm{O}}(dD + D_\oplus)$.*

*Proof.* See Algorithm 15.1. Condition on $\Lambda \in \mathrm{O}(\log n)$ and $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$; both events occur w.h.p. by Lemma 15.1 and Theorem 15.5. By Equation (15.32), we have to compute

$$\mathcal{A}^h(H) = \left(r^V \left(\bigoplus_{\lambda=0}^{\Lambda} P_\lambda (r^V A_\lambda)^d P_\lambda\right)\right)^h r^V x^{(0)}. \tag{15.37}$$

Concerning $P_\lambda$, note that we can evaluate $(P_\lambda y)_{v \in V}$ lazily, i.e., determine whether $(P_\lambda y)_v$ evaluates to $\perp$ or to $y_v$ only if it is accessed. Thus, the total work and depth required by Algorithm 15.1 increase by at most a constant factor due to all applications of $P_\lambda$. Together with the prerequisites, this means that $(r^V A_\lambda P_\lambda)y$ can be determined in $\mathrm{O}(W)$ work and $\mathrm{O}(D)$ depth (line 6), and that evaluating $P_\lambda (r^V A_\lambda)^d P_\lambda y$ sequentially in $d$ requires $\mathrm{O}(dW)$ work and $\mathrm{O}(dD)$ depth (lines 4–8).

The set of summands of $\bigoplus_{\lambda=0}^{\Lambda} P_\lambda (r^V A_\lambda)^d P_\lambda y$ (lines 3–9) can be determined using $\mathrm{O}(\Lambda dW)$ work and $\mathrm{O}(dD)$ depth, since this is independent for each $\lambda$. Performing the aggregation applying the filter (line 10) is possible in $D_\oplus$ depth and $W_\oplus$ work by assumption. We arrive at $\mathrm{O}(\Lambda dW + W_\oplus)$ work and $\mathrm{O}(dD + D_\oplus)$ depth for determining $x^{(i)} = r^V \bigoplus_{\lambda=0}^{\Lambda} P_\lambda (r^V A_\lambda)^d P_\lambda x^{(i-1)}$ from $x^{(i-1)}$.

Sequentially iterating this $h$ times to determine $\mathcal{A}^h(H)$ (lines 2–11) increases work and depth by a factor of $h$, yielding $\mathrm{O}((\Lambda dW + W_\oplus)h)$ work and $\mathrm{O}((dD + D_\oplus)h)$ depth. Computing $r^V x^{(0)}$ (line 1) requires work $W$ and depth $D$ by the prerequisites and does not change the asymptotic complexity accumulated so far. Due to $\Lambda \in \mathrm{O}(\log n)$ and we arrive at $\mathrm{O}((dW \log n + W_\oplus)h)$ work and $\mathrm{O}((dD + D_\oplus)h)$ depth, which is the first claim; $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ yields the second claim. As we only condition on two events that occur w.h.p., this concludes the proof by Lemma 2.7. $\square$

Recall that the motivation for the oracle are subquadratic-work algorithms of polylogarithmic depth that guarantee consistency with the triangle inequality. Specifically, subquadratic refers to near-linear work in $m$ in Chapter 17. Observe that $m$ is not among the parameters of Theorem 15.7 because it is "hidden" in $D$, $W$, $D_\oplus$, and $W_\oplus$, the depth and work required for iterations in $G$ and aggregating their results.

Chapter 16 contains a straightforward application of Theorem 15.7 that demonstrates the usage of the oracle and may serve as example.

# CHAPTER 16

# Parallel Approximate Metrics

Our oracle for MBF-like queries from Chapter 15 permits the efficient simulation of MBF-like algorithms in a graph $H$, which $(1 + \mathrm{o}(1))$-approximates distances in $G$. The entire simulation only has polylogarithmic overhead w.r.t. a single iteration in $G$, provided that we first add an appropriate hop set to $G$. Querying the oracle with the MBF-like algorithm APSP — see Example 14.6 — hence yields a $(1 + \mathrm{o}(1))$-approximate metric of $\operatorname{dist}(\cdot, \cdot, G)$ w.r.t. Definition 12.2; note that respecting the triangle inequality is critical for this problem.

Determining the hop set and querying the oracle with APSP together requires polylogarithmic depth and $\tilde{\mathrm{O}}(nm + n^{2+\varepsilon})$ work, compare Corollary 16.6. In sparse graphs, this is much more work-efficient than the naive approach using $\mathrm{O}(n^3 \log n)$ work — squaring the adjacency matrix $\lceil \log_2 n \rceil$ times — for obtaining $\operatorname{dist}(\cdot, \cdot, G)$ exactly.

Instead of APSP, however, we use MSSP below. MSSP determines $\operatorname{dist}(s, v, G)$ for all $s \in S$ and $v \in V$, where $S \subseteq V$ is a designated set of source nodes, compare Example 14.7. This is a slight generalization of APSP — APSP is MSSP with $S = V$ — that gives us the flexibility to compute the submetric spanned by $S$. It helps, however, to keep APSP and $S = V$ in mind.

Carefully note that we do not reproduce existing distance approximations using hop sets. Although we approximate distances, we require them to form an *approximate metric of* $\operatorname{dist}(\cdot, \cdot, G)$ in the sense of Definition 12.2.

In Theorem 16.1, we formulate the result motivated above independently from a concrete hop-set algorithm. We plug in Cohen's hop set [34] in Theorem 16.2 to obtain specific guarantees.

**Theorem 16.1** (Approximate Multi-Source Shortest Paths). *Let $G = (V, E, \omega)$ be a weighted graph that contains a $(d, \hat{\varepsilon})$-hop set and let $S \subseteq V$ be a set of source nodes. We can w.h.p. compute $A \in (\mathbb{R}_{\geq 0} \cup \{\infty\})^{S \times V}$ such that:*

*(1) $(A_{sv})_{s \in S, v \in V}$ is a subset of the pairwise distances of a $(1 + \hat{\varepsilon})^{\mathrm{O}(\log n)}$-approximate metric of $\operatorname{dist}(\cdot, \cdot, G)$.*

*(2) In particular, $(A_{st})_{s,t \in S}$ is a $(1 + \hat{\varepsilon})^{\mathrm{O}(\log n)}$-approximate submetric of $\operatorname{dist}(\cdot, \cdot, G)$.*

*The computation is possible using $\tilde{\mathrm{O}}(|S|dm)$ work and $\tilde{\mathrm{O}}(d)$ depth.*

*Proof.* MSSP with source nodes $S$, see Example 14.7, is an MBF-like algorithm with at most $|S|$ non-$\infty$ entries in all intermediate node state $x_v^{(i)}$. Hence, an iteration of MSSP incurs $\mathrm{O}(\log n)$ depth and $\mathrm{O}(\delta_v |S| \log n)$ work at node $v$ — where $\delta_v$ is the degree of $v$ — by Lemma 13.3. As this can be done in parallel for all $v \in V$, an entire iteration is possible using

$$D \in \mathrm{O}(\log n) \tag{16.1}$$

depth and

$$W \in \mathrm{O}\left(\sum_{v \in V} \delta_v |S| \log n\right) = \mathrm{O}(|S| m \log n) \subseteq \tilde{\mathrm{O}}(|S|m) \tag{16.2}$$

work. Filtering only initially removes all non-$S$ entries from $x^{(0)}$ and after that remains without effect. So it does not increase the above asymptotic bounds on depth and work. Aggregation over $\Lambda \in \mathrm{O}(\log n)$ vectors, i.e., determining $\bigoplus_{\lambda=0}^{\Lambda} y_\lambda$ for $y_\lambda \in \mathcal{D}$, takes $\mathrm{O}(\log n)$ depth and a work of $\mathrm{O}(\Lambda |S| \log n) = \mathrm{O}(|S| \log^2 n)$ per node by Lemma 13.3. This can be done in parallel for all vertices, resulting in

$$D_\oplus \in \mathrm{O}(\log n) \tag{16.3}$$

depth and

$$W_\oplus \in \mathrm{O}\left(|S| n \log^2 n\right) \subseteq \tilde{\mathrm{O}}\left(|S| n\right) \tag{16.4}$$

work. By Theorem 15.7, we can w.h.p. simulate $\mathrm{SPD}(H)$ iterations of MSSP in $H$ using $\tilde{\mathrm{O}}(dD + D_\oplus) \subseteq \tilde{\mathrm{O}}(d)$ depth and $\tilde{\mathrm{O}}(dW + W_\oplus) \subseteq \tilde{\mathrm{O}}(|S| dm)$ work, as claimed.

It remains to show that we determine a subset of the distances from a $(1+\hat{\varepsilon})^{\mathrm{O}(\log n)}$-approximate metric of $\mathrm{dist}(\cdot,\cdot,G)$. To this end, observe that we determine a subset of $\mathrm{dist}(\cdot,\cdot,H)$ by Theorem 15.7, which is a metric and, by Theorem 15.5, $(1+\hat{\varepsilon})^{\mathrm{O}(\log n)}$-approximates $\mathrm{dist}(\cdot,\cdot,G)$. □

The next step is to plug Cohen's hop-set algorithm [34] into Theorem 16.1 to obtain (a subset of the distances of) a $(1+\mathrm{o}(1))$-approximate metric of $\mathrm{dist}(\cdot,\cdot,G)$ as promised above.

**Theorem 16.2** ((1+o(1))-Approximate Multi-Source Shortest Paths)**.** *Given a weighted graph $G = (V, E, \omega)$, a constant $0 < \varepsilon \leq \frac{1}{2}$, and a set of source nodes $S \subseteq V$, we can w.h.p. compute $A \in (\mathbb{R}_{\geq 0} \cup \{\infty\})^{S \times V}$ such that:*

*(1) $(A_{sv})_{s \in S, v \in V}$ is a subset of the pairwise distances of a $(1+1/\operatorname{polylog} n)$-approximate metric of $\mathrm{dist}(\cdot,\cdot,G)$.*

*(2) In particular, $(A_{st})_{s,t \in S}$ is a $(1+1/\operatorname{polylog} n)$-approximate submetric of $\mathrm{dist}(\cdot,\cdot,G)$.*

*The computation takes $\tilde{\mathrm{O}}(|S|(m + n^{1+\varepsilon}) + m^{1+\varepsilon})$ work and $\operatorname{polylog} n$ depth.*

*Proof.* W.h.p. augment $G$ with a $(d, 1/\operatorname{polylog} n)$-hop set using $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work and $\operatorname{polylog} n$ depth with $d \in \operatorname{polylog} n$ using Cohen's hop-set construction [34]. The resulting graph $G' = (V, E', \omega')$ has $m' := |E'| \in \tilde{\mathrm{O}}(m + n^{1+\varepsilon})$ edges, where $E \subseteq E'$ and $\omega'(e) = \omega(e)$ for all $e \in E$.

Applying Theorem 16.1 to $G'$ yields an additional $\tilde{\mathrm{O}}(d)$ depth and $\tilde{\mathrm{O}}(|S| dm')$ work. Together with the hop-set construction, we obtain $\operatorname{polylog} n$ depth and

$$\tilde{\mathrm{O}}\left(|S| dm' + m^{1+\varepsilon}\right) \subseteq \tilde{\mathrm{O}}\left(|S|\left(m + n^{1+\varepsilon}\right) + m^{1+\varepsilon}\right) \tag{16.5}$$

work as claimed.

The determined distances are from an approximate metric by Theorem 16.1. Using Theorem 15.5 and Equation (15.19), it follows that the metric w.h.p. $(1+1/\operatorname{polylog} n)$-approximates $\mathrm{dist}(\cdot,\cdot,G)$ as claimed. □

With the sparsifier of Baswana and Sen [10], we obtain a different work–approximation trade-off, see Theorem 16.4. In fact, we are near-optimal in terms of work, due to the trivial lower bound of $\Omega(|S|n)$ for writing down the result of MSSP. Before fleshing out the theorem, however, we state the properties of the sparsifier of Baswana and Sen in our model of computation.

Given an integer $k \in \mathbb{N}$, Baswana and Sen [10] show how to compute a $(2k-1)$-spanner of $G = (V, E, \omega)$, i.e., $E_S \subseteq E$ such that $G_S := (V, E_S, \omega)$ fulfills, for all $v, w \in V$,

$$\operatorname{dist}(v, w, G) \leq \operatorname{dist}(v, w, G_S) \leq (2k-1) \operatorname{dist}(v, w, G). \tag{16.6}$$

In expectation, we obtain $|E_S| \in \mathrm{O}(kn^{1+1/k})$. We summarize the original paper [10] with the help of Appendix A of Becker et al. [12].

The algorithm works in $k$ iterations and maintains a clustering of the nodes. In order to assess the depth and work of the algorithm, observe that the bulk of the algorithm happens at the nodes: In each iteration, each cluster is *marked* with probability $n^{-1/k}$ and all nodes in unmarked clusters (1) ignore, for each incident cluster, all but the cheapest edge leading into that cluster, (2) sort the remaining edges by weight, and (3) add the lightest $\ell$ of these edges, where the $\ell$-th edge is the first that is incident to a marked cluster, to $E_S$ and join the corresponding cluster (if there is no incident marked cluster, all remaining edges are added to $E_S$ and the node remains inactive in future iterations).

For a node $v \in V$ of degree $\delta_v$, the above procedure can be implemented by the following steps, each of which is possible using $\mathrm{O}(\log n)$ depth and $\mathrm{O}(\delta_v \log n)$ work: (1) sort cluster–edge pairs lexicographically, (2) discard all tuples that have a predecessor in the same cluster, (3) sort the remaining edges by weight, (4) find the index $\ell$ of the first edge leading to a marked cluster, and (5) mark the first $\ell$ edges as part of $E_S$. Hence, we obtain $\mathrm{O}(\log n)$ depth and $\mathrm{O}(m \log n)$ work per iteration for all nodes, i.e., $\tilde{\mathrm{O}}(k)$ depth and $\tilde{\mathrm{O}}(km)$ work in total. Baswana and Sen argue that $|E_S| \in \mathrm{O}(kn^{1+1/k})$ in expectation [10], it is, however, not hard to show that $|E_S| \in \mathrm{O}(kn^{1+1/k} \log n) \subseteq \tilde{\mathrm{O}}(kn^{1+1/k})$ w.h.p. [12].

**Observation 16.3.** *W.l.o.g., we have $k \in \mathrm{O}(\log n)$ because for $k \geq \log_2 n$, both the stretch and the spanner's number of edges $|E_S|$ grows due to $kn^{1/k} = k2^{\log_2 n/k} > k$.*

We use the spanner of Baswana and Sen [10] to offer an alternative version of Theorem 16.2, essentially reducing the work at the expense of a constant factor regarding the stretch. The result is a constant-factor approximation of MSSP. Note that this result is near-optimal in terms of work due to the trivial lower bound of $\Omega(|S|n)$ for writing down the solution.

**Theorem 16.4** (O(1)-Approximate Multi-Source Shortest Paths)**.** *Given a weighted graph $G = (V, E, \omega)$, a constant $0 < \varepsilon \leq \frac{5}{4}$, and a set of source nodes $S \subseteq V$, we can w.h.p. compute $A \in (\mathbb{R}_{\geq 0} \cup \{\infty\})^{S \times V}$ such that:*

*(1) $(A_{sv})_{s \in S, v \in V}$ is a subset of the pairwise distances of an $\mathrm{O}(1)$-approximate metric of $\operatorname{dist}(\cdot, \cdot, G)$.*

*(2) In particular, $(A_{st})_{s,t \in S}$ is an $\mathrm{O}(1)$-approximate submetric of $\operatorname{dist}(\cdot, \cdot, G)$.*

*The computation takes $\tilde{\mathrm{O}}(|S|n^{1+\varepsilon} + m)$ work and $\operatorname{polylog} n$ depth.*

*Proof.* Choose $\varepsilon' := \sqrt{1+\varepsilon}-1$ and $k := \lceil \varepsilon'^{-1} \rceil$. We first compute a $(2k-1)$-spanner $G_S$ of $G$ and then apply Theorem 16.2 to $G_S$. The details are as follows.

(1) Compute a $(2k-1)$-spanner $G_S = (V, E_S, \omega)$ of $G$. As detailed above and due to Observation 16.3, $G_S$ can be determined using $\tilde{\mathrm{O}}(k) \subseteq \operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(km) \subseteq \tilde{\mathrm{O}}(m)$ work, i.e., within the stated bounds. In particular, this w.h.p. yields $|E_S| \in \tilde{\mathrm{O}}(n^{1+1/k}) \subseteq \tilde{\mathrm{O}}(n^{1+\varepsilon'})$ edges and a stretch that is constant in $n$ and $m$.

(2) Apply Theorem 16.2 to $G_S$ and $\varepsilon'$, observing that

$$0 < \varepsilon' = \sqrt{1+\varepsilon}-1 \leq \sqrt{\frac{9}{4}}-1 = \frac{1}{2}, \tag{16.7}$$

as required. This w.h.p. incurs $\operatorname{polylog} n$ depth and a work of

$$\begin{aligned}
\tilde{\mathrm{O}}\left(|S|\left(|E_S| + n^{1+\varepsilon'}\right) + |E_S|^{1+\varepsilon'}\right) &= \tilde{\mathrm{O}}\left(|S|\left(n^{1+\varepsilon'} + n^{1+\varepsilon'}\right) + n^{(1+\varepsilon')^2}\right) && (16.8)\\
&= \tilde{\mathrm{O}}\left(|S|n^{1+\varepsilon'} + n^{1+\varepsilon}\right) && (16.9)\\
&\subseteq \tilde{\mathrm{O}}\left(|S|n^{1+\varepsilon}\right). && (16.10)
\end{aligned}$$

Together, we obtain a depth of $\operatorname{polylog} n$ and $\tilde{\mathrm{O}}(|S|n^{1+\varepsilon} + m)$ work. By construction, we compute distances in a metric that has a stretch of $(2k-1)(1+\mathrm{o}(1)) \subseteq \mathrm{O}(\varepsilon^{-1})$, i.e., is constant w.r.t. $n$ and $m$. □

As a last step, observe that the above results allow us to compute $(1+\mathrm{o}(1))$- as well as $\mathrm{O}(1)$-approximate submetrics and metrics of $\operatorname{dist}(\cdot,\cdot,G)$. To obtain an approximate submetric of

$$\begin{aligned}
\operatorname{dist}_S &\colon S \times S \to \mathbb{R}_{\geq 0} \cup \{\infty\} && (16.11)\\
\operatorname{dist}_S(s,t) &:= \operatorname{dist}(s,t,G), && (16.12)
\end{aligned}$$

simply ignore the computed $s$-$v$-distances for all $v \in V \setminus S$. For determining an approximate metric of $\operatorname{dist}_V$ choose $S = V$, turning MSSP into APSP.

**Corollary 16.5** (Approximate Submetric). *Let $G = (V, E, \omega)$ be a weighted graph and $S \subseteq V$ be a set of source nodes. We can w.h.p. compute a*

*(1) $(1+1/\operatorname{polylog} n)$-approximate metric of $\operatorname{dist}_S$, using $\tilde{\mathrm{O}}(|S|(m+n^{1+\varepsilon})+m^{1+\varepsilon})$ work and $\operatorname{polylog} n$ depth, where $0 < \varepsilon \leq \frac{1}{2}$ is an arbitrary constant, and*

*(2) $\mathrm{O}(1)$-approximate metric of $\operatorname{dist}_S$, using $\tilde{\mathrm{O}}(|S|n^{1+\varepsilon}+m)$ work and $\operatorname{polylog} n$ depth, where $0 < \varepsilon \leq \frac{5}{4}$ is an arbitrary constant.*

*In both cases, the resulting metric is represented as a matrix in $(\mathbb{R}_{\geq 0} \cup \{\infty\})^{S \times S}$, i.e., offers constant-time query access.*

*Proof.* Follows from Theorems 16.2 and 16.4 by restricting the output to $(A_{st})_{s,t \in S}$. □

**Corollary 16.6** (Approximate Metric)**.** *Let $G = (V, E, \omega)$ be a weighted graph. Then we can w.h.p. compute a*

*(1)* $(1 + 1/\operatorname{polylog} n)$*-approximate metric of* $\operatorname{dist}(\cdot, \cdot, G)$*, using* $\tilde{O}(n(m + n^{1+\varepsilon}))$ *work and* $\operatorname{polylog} n$ *depth, where* $0 < \varepsilon \leq \frac{1}{2}$ *is an arbitrary constant, and*

*(2)* $O(1)$*-approximate metric of* $\operatorname{dist}(\cdot, \cdot, G)$*, using* $\tilde{O}(n^{2+\varepsilon})$ *work and* $\operatorname{polylog} n$ *depth, where* $0 < \varepsilon \leq \frac{5}{4}$ *is an arbitrary constant.*

*In both cases, the resulting metric is represented as a matrix in* $(\mathbb{R}_{\geq 0} \cup \{\infty\})^{V \times V}$*, i.e., offers constant-time query access.*

*Proof.* Follows from plugging $S = V$ in Corollary 16.5 and observing that in the first claim $m^{1+\varepsilon} \leq m^{3/2} \leq nm$. □

Blelloch et al. show how to construct an FRT tree from a metric using $O(n^2)$ work and $O(\log^2 n)$ depth [21]. Combining this with Corollary 16.6 enables us to w.h.p. construct an FRT tree from a graph $G$ using polylogarithmic depth and $\tilde{O}(n^{2+\varepsilon})$ work, which already improves upon the state of the art. While this does not yield the same FRT tree as when directly embedding $G$ since we "embed an approximation of $\operatorname{dist}(\cdot, \cdot, G)$," it has the same expected asymptotic stretch of $O(\log n)$ due to the constant-factor approximation provided by Corollary 16.6. This can, however, be done more efficiently in sparse graphs: Constructing FRT trees is an MBF-like algorithm and solving the problem using the oracle reduces the work to $\tilde{O}(m^{1+\varepsilon})$, alternatively to $\tilde{O}(m + n^{1+\varepsilon})$ at the expense of increasing the stretch by a factor of $O(\varepsilon^{-1})$; we discuss this in Chapter 17.

# CHAPTER 17

# Parallel FRT Embeddings

This chapter develops our main result (Theorem 17.11, Corollary 17.12, and Corollary 17.13): Given a weighted graph $G$, we can w.h.p. sample a tree embedding of $G$ with expected stretch $\mathrm{O}(\log n)$ using $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work, where $0 < \varepsilon \leq \frac{1}{2}$ is an arbitrary constant. This brings our work close to the sequential state of the art, the algorithm of Blelloch et al. which w.h.p. requires $\mathrm{O}(m \log n)$ time [20]. We do this by sampling an FRT tree [50] in $H$. Alternatively, we can reduce the work to $\tilde{\mathrm{O}}(m + n^{1+\varepsilon})$ at the expense of an expected stretch of $\mathrm{O}(\varepsilon^{-1} \log n)$ for any $0 < \varepsilon \leq \frac{5}{4}$ using the sparsifier of Baswana and Sen [10].

As discussed in Chapter 16, we can use Corollary 16.6 to explicitly construct a metric that $\mathrm{O}(1)$-approximates $\operatorname{dist}(\cdot, \cdot, G)$ using polylogarithmic depth and $\tilde{\mathrm{O}}(n^{2+\varepsilon})$ work; sampling an FRT tree on that metric is possible within the same bounds of work and depth, as demonstrated by Blelloch et al. [21]. The challenge is to reduce the work to $\tilde{\mathrm{O}}(m^{1+\varepsilon})$. To this end, we show that we can collect the information required to construct an FRT tree — the LE lists of $H$ — with an MBF-like algorithm. This algorithm serves as a query that can be directly answered by the oracle (Theorem 15.7) instead of taking the detour of explicitly determining an approximate metric. A single iteration of collecting LE lists has $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m)$ work w.h.p. The oracle can simulate the entire algorithm in $H$ with only a polylogarithmic overhead w.r.t. one such iteration. Since $\operatorname{dist}(\cdot, \cdot, H)$ is a $(1 + \mathrm{o}(1))$-approximation of $\operatorname{dist}(\cdot, \cdot, G)$, we only need to add Cohen's hop set [34] to $G$ to achieve our result.

We give a formal definition of metric embeddings in general and the FRT embedding in particular in Section 17.1, proceed to show that the underlying algorithm is MBF-like and that all intermediate steps are sufficiently efficient in terms of depth and work in Sections 17.2 and Section 17.3, and present our main result in Section 17.4. Section 17.5 describes how to retrieve the original paths in $G$ that correspond to the edges of the sampled FRT tree and in Section 17.6 we discuss how to construct a tree embedding without Steiner nodes that guarantees the same asymptotic stretch.

## 17.1 FRT Embeddings from LE Lists

We use this section to introduce the random distribution over metric tree embeddings of Fakcharoenphol, Rao, and Talwar (FRT), which has expected stretch $\mathrm{O}(\log n)$ [50]. Furthermore, we introduce LE lists and show that computing them directly yields an FRT tree. This establishes the efficient computation of LE lists as the main objective for the remaining part of this chapter. We begin with a formal definition of metric embeddings.

**Definition 17.1** (Metric Embedding)**.** *Let $G = (V, E, \omega)$ be a graph. A* metric embedding *of $G$ is a graph $G' = (V', E', \omega')$ such that $V \subseteq V'$ and*

$$\forall v, w \in V\colon \quad \operatorname{dist}(v, w, G) \le \operatorname{dist}(v, w, G') \le \alpha \operatorname{dist}(v, w, G), \tag{17.1}$$

*where $\alpha \in \mathbb{R}_{\ge 1}$ is referred to as* stretch. *If $G'$ is a tree, we refer to it as* metric tree embedding. *For a random distribution of metric embeddings, we require* $\operatorname{dist}(v, w, G) \le \operatorname{dist}(v, w, G')$ *and define the* expected stretch *as*

$$\alpha := \max_{v \ne w \in V} \mathbb{E}\left[\frac{\operatorname{dist}(v, w, G')}{\operatorname{dist}(v, w, G)}\right]. \tag{17.2}$$

We continue with an introduction to the FRT embedding. For a full description, we refer to the original paper [50]; as an alternative summary, which focuses on LE lists and is thus closer to our perspective, we recommend Ghaffari and Lenzen [65]. Informally, the idea behind FRT trees, compare Figure 17.1, is the following. Consider a graph $G = (V, E, \omega)$ and one of its vertices $v \in V$. Furthermore, fix some order of the vertices; in the context of such an order, we write $v \le w$ ($v < w$) if $v$ is ordered (strictly) before $w$. Begin with a ball around $v$, starting at a radius so small that it only contains $v$. Iteratively double the ball's radius until it spans the entire graph; we demonstrate this for all nodes in Figure 17.1(b). For each iteration, consider the minimum vertex in the corresponding ball around $v$. Initially this is $v$, however, the minimum vertex changes with increasing radius. Eventually, the minimum is $\min V$; this must happen at the latest when the ball around $v$ contains all vertices. We obtain a sequence $v = v_0, \dots, v_k = \min V$. That sequence is a leaf in the FRT tree and we identify it with $v$. The parent of the leaf is obtained by discarding the first item of the sequence, its grandparent by discarding the first two, its great-grandparent by discarding the first three, and so on, compare Figure 17.1(c). The last item of every sequence is $\min V$ which is the root of the FRT tree. Accounting for the exponential growth of the balls with exponentially increasing edge weights, this completes the construction of the FRT tree. Observe that each $v \in V$ is associated with one and only one leaf in the tree.

Formally, consider a graph $G = (V, E, \omega)$, a real number $1 \le \beta < 2$, and an order of $V$. Denote by $\omega_{\min} := \min\{\omega(e) \mid e \in E\}$ and $\omega_{\max} := \max\{\omega(e) \mid e \in E\}$ the minimum and maximum edge weight, respectively. Let $\mathrm{B}(v, r) := \{w \in V \mid \operatorname{dist}(v, w, G) \le r\}$ denote the *ball of radius $r$ around $v \in V$*. For $v \in V$, define the sequence $(v_0, \dots, v_k)$ by

$$(v_i)_{i \in \{0,\dots,k\}} := \min \mathrm{B}\left(v, 2^{i-1}\beta\, \omega_{\min}\right), \tag{17.3}$$

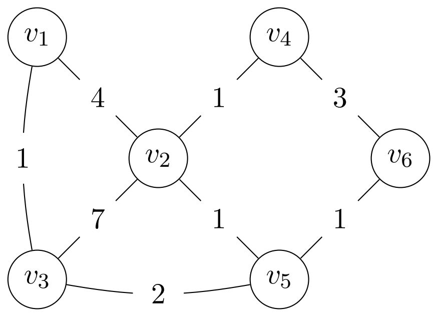
choosing

$$k := 1 + \left\lceil \log_2 \frac{\max_{v \in V} \operatorname{dist}(v, \min V, G)}{\beta\, \omega_{\min}} \right\rceil. \tag{17.4}$$

Observe that the radii are chosen such that, for all $v \in V$, we have $v_0 = v$ and $v_k = \min V$, because, due to $\beta/2 < 1$,

$$2^{0-1}\beta\, \omega_{\min} < \omega_{\min} \text{ and} \tag{17.5}$$

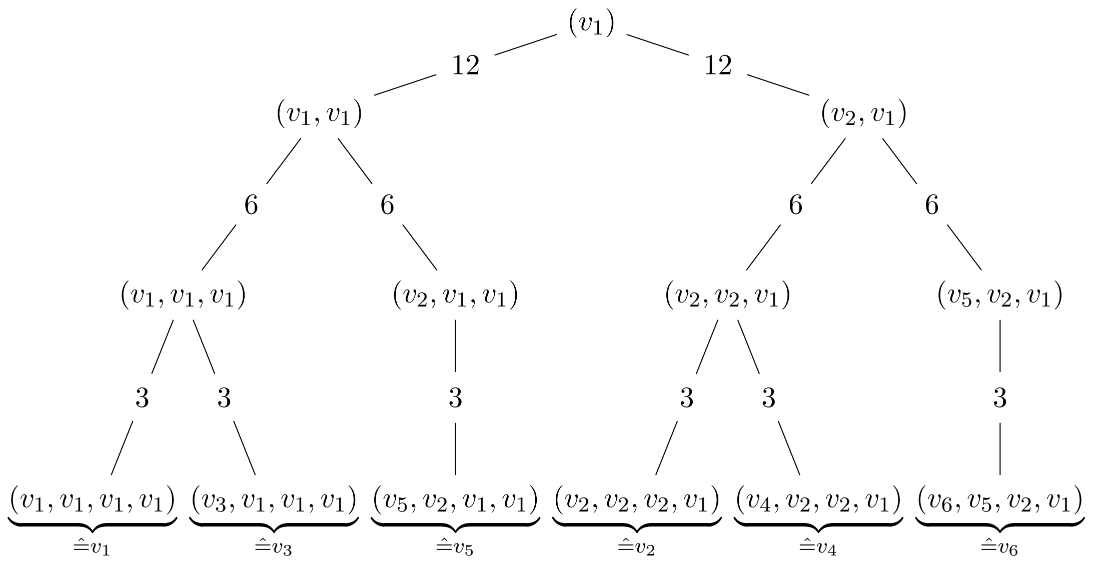
$$2^{k-1}\beta\, \omega_{\min} \ge \max_{v \in V} \operatorname{dist}(v, \min V, G). \tag{17.6}$$



**(a)** A weighted graph $G = (V, E, \omega)$ with $\omega_{\min} = 1$ and $\max_{v \in V} \operatorname{dist}(v, v_1, G) = 5$, resulting in $k = 3$ for $\beta = 1.5$.

| $i$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $2^{i-1}\beta\,\omega_{\min}$ | 0.75 | 1.5 | 3 | 6 |
| $\mathrm{B}(v_1, 2^{i-1}\beta\,\omega_{\min})$ | $\{\underline{v_1}\}$ | $\{\underline{v_1}, v_3\}$ | $\{\underline{v_1}, v_3, v_5\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ |
| $\mathrm{B}(v_2, 2^{i-1}\beta\,\omega_{\min})$ | $\{\underline{v_2}\}$ | $\{\underline{v_2}, v_4, v_5\}$ | $\{\underline{v_2}, v_3, v_4, v_5, v_6\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ |
| $\mathrm{B}(v_3, 2^{i-1}\beta\,\omega_{\min})$ | $\{\underline{v_3}\}$ | $\{\underline{v_1}, v_3\}$ | $\{\underline{v_1}, v_2, v_3, v_5, v_6\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ |
| $\mathrm{B}(v_4, 2^{i-1}\beta\,\omega_{\min})$ | $\{\underline{v_4}\}$ | $\{\underline{v_2}, v_4\}$ | $\{\underline{v_2}, v_4, v_5, v_6\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ |
| $\mathrm{B}(v_5, 2^{i-1}\beta\,\omega_{\min})$ | $\{\underline{v_5}\}$ | $\{\underline{v_2}, v_5, v_6\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ |
| $\mathrm{B}(v_6, 2^{i-1}\beta\,\omega_{\min})$ | $\{\underline{v_6}\}$ | $\{\underline{v_5}, v_6\}$ | $\{\underline{v_2}, v_3, v_4, v_5, v_6\}$ | $\{\underline{v_1}, v_2, v_3, v_4, v_5, v_6\}$ |

**(b)** Exponentially growing balls around the vertices of $G$ with $\beta = 1.5$.



**(c)** The resulting FRT tree.

**Figure 17.1:** We construct the FRT tree of the graph $G$ in (a) w.r.t. the node order $v_1 < \cdots < v_6$ and $\beta = 1.5$. The exponentially growing balls around each vertex are given in (b). The minima (underlined) in the balls around $v_i$ determine a sequence of nodes; discarding the first element yields the parent in the corresponding FRT tree (c).

As $\max_{v \in V} \operatorname{dist}(v, \min V, G)$ is bounded from above by $n\,\omega_{\max}$ and $\omega_{\max} / \omega_{\min} \in \operatorname{poly} n$ by assumption, we obtain

$$k \in \mathrm{O}(\log n). \tag{17.7}$$

With the terminology in place, we proceed to a formal definition FRT trees, embeddings, and distributions.

**Definition 17.2** (FRT Tree, Embedding, and Distribution [50])**.** *Let $G = (V, E, \omega)$ be a graph and $1 \leq \beta < 2$ a real number. Fix an order on $V$. The* Fakcharoenphol, Rao, and Talwar (FRT) tree *of $G$ w.r.t. $\beta$ and the node order is $T := (V_T, E_T, \omega_T)$ with*

$$V_T := \{(v_i, \dots, v_k) \mid v \in V, 0 \leq i \leq k\}\,, \tag{17.8}$$

$$E_T := \{\{(v_i, v_{i+1}, \dots, v_k), (v_{i+1}, \dots, v_k)\} \mid v \in V, 0 \leq i < k\}\,,\ \textit{and} \tag{17.9}$$

$$\omega_T(\{(v_i, v_{i+1}, \dots, v_k), (v_{i+1}, \dots, v_k)\}) := 2^{i+1}\beta\,\omega_{\min} \tag{17.10}$$

*for $k$ from Equation* (17.4)*. When identifying each $v \in V$ with $(v_0, \dots, v_k)$, $T$ is a metric tree embedding of $G$, referred to as* FRT embedding*. The random distribution of FRT trees w.r.t. $G$ defined by choosing uniformly at random $1 \leq \beta < 2$ and, also uniformly at random, a permutation of $V$ as node order is referred to as the* FRT distribution.

**Remark 17.3.** *Fakcharoenphol et al. in the conference version of [50], Khan et al. [81], and Ghaffari and Lenzen [65] choose $1 \leq \beta < 2$ uniformly at random. In the journal version, however, Fakcharoenphol et al. [50] choose $1 \leq \beta \leq 2$ randomly w.r.t. the probability density function $p(x) = \frac{1}{x \ln 2}$. This choice does not have an impact on the expected asymptotic stretch of the FRT distribution. As the random distribution for $\beta$ can be easily adapted in our algorithms and for the sake of presentation, we only explicitly state the option of choosing $\beta$ uniformly at random in the following.*

For an arbitrary graph $G$, the FRT distribution has expected stretch $\mathrm{O}(\log n)$ [50]. Bartal established that this expected stretch is asymptotically optimal [8].

The FRT embedding critically depends on the subtractive form of the triangle inequality, compare the reasoning leading up to Theorem 2 of Fakcharoenphol et al. [50]. Hence, as detailed in Chapter 15, we need the oracle — Theorem 15.7 — to sample from the FRT distribution in only polylogarithmic depth, as we cannot afford the computation of exact distances. More precisely, we sample from the FRT distribution for the graph $H$ introduced in Chapter 15 instead of $G$. Since $H$ is an embedding of $G$ with a stretch of $1 + \mathrm{o}(1)$, we w.h.p. obtain a tree embedding of $G$ that has an expected stretch in $(1 + \mathrm{o}(1))\,\mathrm{O}(\log n) = \mathrm{O}(\log n)$.

In order to apply the oracle, we need to show that the information required to sample from the FRT distribution can be obtained by an MBF-like algorithm over $\mathcal{S}_{\min,+}$. To this end, we employ the approach of Khan et al., who use LE lists as a suitable representation for sampling an FRT tree [81]. Given a graph $G = (V, E, \omega)$ and a node order, the *Least Element (LE) list* — a concept first introduced by Cohen [33] — of $v \in V$ is

$$\{(w, \operatorname{dist}(v, w, G)) \mid w \in V, w = \min \mathrm{B}\,(v, \operatorname{dist}(v, w, G))\}\,. \tag{17.11}$$

Essentially, $v$ learns, for every distance $d$, the smallest node within distance at most $d$, i.e., $\min\{w \in V \mid \operatorname{dist}(v, w, G) \leq d\}$.

```
input: A graph $G = (V, E, \omega)$
output: An FRT tree of $G$
1: sample $1 \leq \beta < 2$ uniformly at random
2: sample a node order uniformly at random
3: for each $v \in V$ do
4:     $L_v \leftarrow$ LE list of $v$
5:     determine ancestors $(v_0, v_1, v_2, \dots, v_k)$, $(v_1, v_2, \dots, v_k)$, $\dots$, $(v_k)$ of $v$ from $L_v$
6: end for
7: return FRT tree assembled from individual ancestor lists
```

**Algorithm 17.1:** FRT embedding via LE lists, adapted from Khan et al. [81].

Observe that the FRT tree is uniquely determined by the LE lists and $\beta$. The LE list of $v$ allows the easy recovery of $\min \mathrm{B}(v, r)$ for any radius $r$. Regarding the FRT tree, we require that information for a small set of radii, i.e., the $k + 1$ exponentially growing radii used in Equation (17.3).

In a nutshell, Khan et al. demonstrate that an FRT tree can be sampled by using the high-level approach outlined in Algorithm 17.1: (1) Sample $\beta$ and a random node order, (2) determine the according LE lists, and (3) construct the resulting FRT tree [81]. We follow the same overall approach. Khan et al. w.h.p. need $\mathrm{O}(\mathrm{SPD}(G) \log n)$ rounds in the CONGEST model[1] to determine the exact distances in $G$ needed for the LE lists. $\Omega(\mathrm{SPD}(G))$ rounds imply $\Omega(\mathrm{SPD}(G))$ depth in our model of computation, which is a problem as $\mathrm{SPD}(G) = n - 1$ is possible. Hence, as motivated above, we circumnavigate the obstacle by instead using the graph $H$ with $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ via the oracle (Theorem 15.7). To this end, we show that the computation of LE lists is an MBF-like algorithm over $\mathcal{S}_{\min,+}$ that w.h.p. requires polylog $n$ depth and $\tilde{\mathrm{O}}(m)$ work per iteration.

Showing that determining LE lists is an efficient MBF-like algorithm is the main part of this chapter. However, we also need to show that constructing the FRT tree from the individual LE lists — lines 5 and 7 of Algorithm 17.1 — is straightforward if the LE lists are not too long. The assumption of short LE lists holds w.h.p., which we establish in Section 17.3.

**Lemma 17.4.** *Given LE lists of length* $\mathrm{O}(\log n)$ *for all vertices, the corresponding FRT tree can be determined using* $\mathrm{O}(n \log^3 n)$ *work and* $\mathrm{O}(\log^2 n)$ *depth.*

*Proof.* Determining $\omega_{\min}$ and $k$ is straightforward at this complexity, as is sorting of each node's LE list in ascending order w.r.t. distance. Note that, in each resulting list of node–distance pairs, the nodes are strictly decreasing in terms of the random node order and that each list ends with an entry for the minimal node $\min V$.

For each node $v$ and entry $(\mathrm{dist}(v, w, G), w)$ in its list in parallel, we determine the values of $i \in \{0, \dots, k\}$ such that $w = \min \mathrm{B}(v, 2^{i-1} \beta\, \omega_{\min})$ is the smallest node within distance $2^{i-1} \beta\, \omega_{\min}$ of $v$. This is done by reading the distance value $d'$ of the next entry of the list — using $d' = \infty$ if $(\mathrm{dist}(v, w, G), w)$ is the last entry — and writing to memory

[1]The $\log n$ factor is owed to the CONGEST model. LE lists contain $\mathrm{O}(\log n)$ elements w.h.p., each of $\mathrm{O}(\log n)$ bits, as they contain a node ID. Hence, it takes $\mathrm{O}(\log n)$ rounds in the CONGEST model to communicate an LE list to a neighboring node.

$v_i := w$ for each $i$ satisfying that $\mathrm{dist}(v, w, G) \leq 2^{i-1}\beta\,\omega_{\min} < d'$. By Equation (17.7), this has depth $\mathrm{O}(\log n)$. The total work is $\mathrm{O}(n \log^2 n)$.

Observe that the above procedure yields the sequence $(v_0, \dots, v_k)$ for each $v \in V$, i.e., the leafs of the FRT tree. Recall that the ancestors of $(v_0, \dots, v_k)$ are determined by its $k$ suffixes. It remains to remove duplicates wherever nodes share a parent. To this end, we sort the list of $(k+1)n \in \mathrm{O}(n \log n)$ suffixes — each with $\mathrm{O}(\log n)$ entries — lexicographically, requiring $\mathrm{O}(n \log^3 n)$ work and depth $\mathrm{O}(\log^2 n)$, as comparing two suffixes requires depth and work $\mathrm{O}(\log n)$. Then duplicates can be removed by comparing each key to its successor in the sorted sequence, taking another $\mathrm{O}(n \log^2 n)$ work and $\mathrm{O}(\log n)$ depth. Overall, we spend $\mathrm{O}(n \log^3 n)$ work at $\mathrm{O}(\log^2 n)$ depth, as claimed. □

Note that tree edges and their weights are encoded implicitly: The parent of each node is given by removing the first node from the sequence, the level of a node is given by the length of the sequence representing it, and the weight of the according edge is uniquely determined by the node levels, $\beta$ and $\omega_{\min}$. If required, it is thus trivial to determine, e.g., an adjacency list within the work and depth stated in Lemma 17.4.

## 17.2 Computing LE Lists is MBF-like

Picking $\beta$ (line 1 of Algorithm 17.1) is trivial. Furthermore, it is a well-known fact [65, 81] that choosing a random order of the nodes (line 2 of Algorithm 17.1) can be done w.h.p. by assigning to each node a string of $\mathrm{O}(\log n)$ uniformly and independently chosen random bits. Hence, in the following, we assume these steps to be completed, resulting in a total order of the vertices. It remains to show how to efficiently compute LE lists.

We establish that LE lists can be computed by an MBF-like algorithm, see Chapter 13, using the parameters in Definition 17.5. To this end, we need to show that Equations (17.12) and (17.13) define a representative projection and a congruence relation, which we do in Lemma 17.7.

**Definition 17.5.** *For constructing LE lists, use the semiring $\mathcal{S} = \mathcal{S}_{\min,+}$ and the distance map $\mathcal{M} = \mathcal{D}$ from Definition 13.1 as zero-preserving semimodule. For all $x \in \mathcal{D}$, define*

$$r(x)_v := \begin{cases} \infty & \exists w < v\colon\ x_w \leq x_v \text{ and} \\ x_v & \text{otherwise, and} \end{cases} \tag{17.12}$$

$$x \sim y \quad :\Leftrightarrow \quad r(x) = r(y) \tag{17.13}$$

*as representative projection and congruence relation, respectively. We use $x^{(0)} \in \mathcal{D}^V$ with*

$$x^{(0)}_{vw} := \begin{cases} 0 & \text{if } v = w \text{ and} \\ \infty & \text{otherwise} \end{cases} \tag{17.14}$$

*as initialization.*

Hence, $r(x)$ is the LE list of $v \in V$ if $x_w = \mathrm{dist}(v, w, G)$ for all $w \in V$. We say that a node–distance pair $(v, d)$ *dominates* $(v', d')$ if and only if $v < v'$ and $d \leq d'$; in the context of $x \in \mathcal{D}$, we say that $x_w$ *dominates* $x_v$ if and only if $(w, x_w)$ dominates $(v, x_v)$. The filter in Equation (17.12) discards exactly the dominated entries in $x$.

We prepare the proof that LE lists can be retrieved by an MBF-like algorithm with the following lemma. It states that filtering keeps the relevant information: If a node–distance pair is dominated by an entry in a distance map, the filtered distance map also contains a — possibly different — dominating entry.

**Lemma 17.6.** *Let $x, y \in \mathcal{D}$, $v \in V$, and $s \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ be arbitrary. Then*

$$\exists w < v \colon x_w \leq s \quad \Leftrightarrow \quad \exists w < v \colon r(x)_w \leq s \tag{17.15}$$

*Proof.* Observe that the necessity "$\Leftarrow$" is trivial. As for sufficiency "$\Rightarrow$," suppose that there is $w < v$ such that $x_w \leq s$. If $r(x)_w = x_w$, we are done. Otherwise, there must be some $u < w < v$ satisfying $x_u \leq x_w \leq s$ by definition of $r$, i.e., some $x_u$ dominating $x_w$. Since $|V|$ is finite and domination induces a partial order, an inductive repetition of the argument reveals that there is some $w' < v$ with $r(x)_{w'} = x_{w'} \leq s$. □

Equipped with this lemma, we can prove that $\sim$ is a congruence relation on $\mathcal{D}$ with representative projection $r$.

**Lemma 17.7.** *The equivalence relation $\sim$ from Equation* (17.13) *is a congruence relation and $r$ from Equation* (17.12) *is a representative projection w.r.t. $\sim$.*

*Proof.* Trivially, $r$ is a projection, i.e., $r^2(x) = r(x)$ for all $x \in \mathcal{D}$. By Lemma 13.8, it hence suffices to show that (13.12) and (13.13) hold, which we show in one step each. To this end, let $v \in V$ as well as $s \in \mathcal{S}_{\min,+}$ be arbitrary and $x, x', y, y' \in \mathcal{D}$ such that $r(x) = r(x')$ and $r(y) = r(y')$ throughout the proof.

(1) Concerning (13.12), observe that, trivially,

$$\exists w < v \colon x_w \leq x_v \quad \Leftrightarrow \quad \exists w < v \colon s + x_w \leq s + x_v. \tag{17.16}$$

Hence, we have $r(sx) = sr(x)$ and it follows that $r(sx) = sr(x) = sr(x') = r(sx')$.

(2) Regarding (13.13), we show that

$$r(x \oplus y) = r(r(x) \oplus r(y)) \tag{17.17}$$

which implies (13.13) due to $r(x \oplus y) = r(r(x) \oplus r(y)) = r(r(x') \oplus r(y')) = r(x' \oplus y')$. First observe that $(x \oplus y)_v$ is dominated if and only if

$$\begin{aligned}
& \exists w < v \colon && (x \oplus y)_w \leq (x \oplus y)_v && (17.18)\\
\Leftrightarrow \; & \exists w < v \colon && \min\{x_w, y_w\} \leq (x \oplus y)_v && (17.19)\\
\Leftrightarrow \; & \exists w < v \colon && x_w \leq (x \oplus y)_v \vee y_w \leq (x \oplus y)_v && (17.20)\\
\overset{(17.15)}{\Leftrightarrow} \; & \exists w < v \colon && r(x)_w \leq (x \oplus y)_v \vee r(y)_w \leq (x \oplus y)_v && (17.21)\\
\Leftrightarrow \; & \exists w < v \colon && \min\{r(x)_w, r(y)_w\} \leq (x \oplus y)_v && (17.22)\\
\Leftrightarrow \; & \exists w < v \colon && (r(x) \oplus r(y))_w \leq (x \oplus y)_v. && (17.23)
\end{aligned}$$

In order to show (17.17), we distinguish two cases.

**Case 1** If $(x \oplus y)_v$ is dominated, we obtain $r(x \oplus y)_v = \infty$ by definition of $r$. Additionally, we know that

$$(r(x) \oplus r(y))_v = \min\{r(x)_v, r(y)_v\} \geq \min\{x_v, y_v\} = (x \oplus y)_v. \tag{17.24}$$

Hence, $(r(x) \oplus r(y))_v$ must be dominated due to (17.23). We conclude that $r(r(x) \oplus r(y))_v = \infty = r(x \oplus y)_v$.

**Case 2** If $(x \oplus y)_v$ is not dominated, suppose w.l.o.g. that $x_v \leq y_v$; the other case is symmetric. We obtain

$$r(x \oplus y)_v = (x \oplus y)_v = x_v. \tag{17.25}$$

Furthermore, the negation of (17.23) holds, i.e.,

$$\forall w < v\colon \quad (x \oplus y)_v < (r(x) \oplus r(y))_w, \tag{17.26}$$

and we conclude

$$\forall w < v\colon \quad x_v \overset{(17.25)}{=} (x \oplus y)_v \overset{(17.26)}{<} (r(x) \oplus r(y))_w \leq r(x)_w. \tag{17.27}$$

Plugging $\forall w < v\colon x_v < r(x)_w$ into Equation (17.15), flipping the quantifier and using $s = x_v$, yields

$$\forall w < v\colon x_v < r(x)_w \overset{(17.15)}{\Leftrightarrow} \underbrace{\forall w < v\colon x_v < x_w}_{x_v \text{ is not dominated}} \Leftrightarrow x_v = r(x)_v, \tag{17.28}$$

where the last step follows from recalling that $r$ discards exactly the entries that are dominated. Next, observe that

$$(r(x) \oplus r(y))_v = \min\{r(x)_v, r(y)_v\} \overset{(17.28)}{=} \min\{x_v, r(y)_v\} = x_v; \tag{17.29}$$

the last step follows from $r(y)_v \geq y_v$ and our initial assumption that w.l.o.g. $x_v \leq y_v$. This yields

$$\forall w < v\colon (r(x) \oplus r(y))_v \overset{(17.29)}{=} x_v \overset{(17.25)}{=} (x \oplus y)_v \overset{(17.26)}{<} (r(x) \oplus r(y))_w. \tag{17.30}$$

In particular, $(r(x) \oplus r(y))_v$ is not dominated. Hence, $r$ does not discard it and we obtain

$$r(r(x) \oplus r(y))_v = (r(x) \oplus r(y))_v \overset{(17.29)}{=} x_v \overset{(17.25)}{=} r(x \oplus y)_v. \tag{17.31}$$

As $v$ is arbitrary, this concludes the case.

The case distinction establishes that Equation (17.17) holds which, as argued above, implies (13.13).

Together, the claim follows. □

Having established that LE lists can be determined by an MBF-like algorithm allows us to apply the machinery developed in Chapters 13 and 15. Next, we establish that LE-list computations can be performed efficiently, which we show by bounding the length of LE lists.

## 17.3 Computing LE Lists is Efficient

Our course of action is to use the oracle theorem — Theorem 15.7 — for an efficient computation of LE lists. As detailed in Chapter 15, the oracle is our strategy to balance the need for a polylogarithmic SPD — to obtain an algorithm of polylogarithmic depth — while at the same time not violating the triangle inequality and guaranteeing a good distance approximation. The purpose of this section is to prepare the lemmas required to apply Theorem 15.7. We stress that the key challenge is to perform each iteration in $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m)$ work; this allows us to collect the LE lists in $H$ within the same bounds of work and depth — up to a small overhead in work caused by the additional edges of Cohen's hop set [34] — because the oracle induces only a polylogarithmic overhead w.r.t. both parameters.

To this end, we first establish the length of intermediate LE lists to be logarithmic w.h.p. in Lemma 17.8. We remark that LE lists are known to have length $\mathrm{O}(\log n)$ w.h.p. throughout intermediate computations [65, 81]; however, these results assume that LE lists contain $h$-hop distances. Lemma 17.8 uses the same key argument but is more general, since it makes no assumption about $x$ except for its independence of the random node order; we need the more general statement due to our decomposition of $A_H$, compare Chapter 15.

Recall that by $|x|$ we denote the number of non-$\infty$ entries of $x \in \mathcal{D}$ and that we only need to keep the non-$\infty$ entries in memory. Lemma 17.8 shows that any LE list $r(x) \in \mathcal{D}$ has length $|r(x)| \in \mathrm{O}(\log n)$ w.h.p., provided that $x$ does not depend on the random node order. Observe that the lemma is quite powerful, as it suffices that there is *any* $x' \sim x$ that does not depend on the random node order: as it holds that $r(x) = r(x')$, we have $|r(x)| = |r(x')| \in \mathrm{O}(\log n)$ w.h.p.

**Lemma 17.8.** *Consider $x \in \mathcal{D}$. If $x$ and the random order of the nodes are independent, $|r(x)| \in \mathrm{O}(\log n)$ w.h.p.*

*Proof.* Order the non-$\infty$ values of $x$ by ascending distance, breaking ties independently of the random node order. Denote for $i \in \{1, \dots, |x|\}$ by $v_i \in V$ the $i$-th node w.r.t. this order, i.e., $x_{v_i}$ is the $i$-th smallest entry in $x$. Furthermore, denote by $X_i$ the indicator variable that is 1 if $v_i < v_j$ for all $j \in \{1, \dots, i-1\}$ and 0 otherwise. As the node order and $x$ are independent, we obtain $\mathbb{E}[X_i] = 1/i$. For $X := \sum_{i=1}^{|x|} X_i$, this implies

$$\mathbb{E}[X] = \sum_{i=1}^{|x|} \frac{1}{i} \leq \sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n). \tag{17.32}$$

Observe that $\{X_1, \dots, X_{i-1}\}$ and $X_i$ are independent, as whether $v_i < v_j$ for all $j < i$ is independent of the internal order of the set $\{v_1, \dots, v_{i-1}\}$.

We claim that $\{X_1, \dots, X_{|x|}\}$ are independent and show it using that $\{X_1, \dots, X_k\}$ are independent if $\mathbb{P}[(X_1, \dots, X_k) = (x_1, \dots, x_k)] = \prod_{i=1}^{k} \mathbb{P}[X_i = x_i]$ for any possible assignment $(x_1, \dots, x_k) \in \mathbb{B}^k$. This trivially holds for $k = 1$. Assuming that $\{X_1, \dots, X_k\}$ are independent as an induction hypothesis, the independence of $\{X_1, \dots, X_{k+1}\}$ follows from

$$\begin{aligned}
&\mathbb{P}[(X_1,\dots,X_{k+1})=(x_1,\dots,x_{k+1})] && (17.33)\\
=&\,\mathbb{P}[(X_1,\dots,X_k)=(x_1,\dots,x_k)]\cdot\mathbb{P}[X_{k+1}=x_{k+1}] && (17.34)\\
=&\left(\prod_{i=1}^{k}\mathbb{P}[X_i=x_i]\right)\cdot\mathbb{P}[X_{k+1}=x_{k+1}] && (17.35)\\
=&\prod_{i=1}^{k+1}\mathbb{P}[X_i=x_i], && (17.36)
\end{aligned}$$

where the first step follows from the above observation that $\{X_1,\dots,X_{i-1}\}$ and $X_i$ are independent, the second step from the induction hypothesis, and the third step is trivial.

Applying Chernoff's bound — see Lemma 2.8 and Corollary 2.9 — yields that $X \in \mathrm{O}(\log n)$ w.h.p. As $\mathbb{P}[X=k]=\mathbb{P}[|r(x)|=k]$, this concludes the proof. □

Hence, filtered, possibly intermediate, LE lists $r(x)$ w.h.p. comprise $\mathrm{O}(\log n)$ entries. We proceed to show that under these circumstances, $r(x)$ can be computed efficiently.

**Lemma 17.9.** *Let $x \in \mathcal{D}$ be arbitrary. Then $r(x)$ can be computed using $\mathrm{O}(|r(x)|\log n)$ depth and $\mathrm{O}(|r(x)||x|)$ work.*

*Proof.* We use one iteration per non-$\infty$ entry of $r(x)$. In each iteration, we first identify the smallest vertex $v$ w.r.t. the random node order that is not *discarded,* then copy $x_v$ to $r(x)_v$, and finally mark all entries of $x$ dominated by $x_v$ as discarded. The invariant is that the nodes are moved to $r(x)$ in ascending order w.r.t. the random node permutation; all non-discarded non-$\infty$ entries of $x$ are ordered after those on $r(x)$. This yields $|r(x)|$ iterations as follows:

(1) Initialize $r(x) \leftarrow \perp$. Construct a tournament tree on the non-$\infty$ elements of $x$ and identify its leaves with their indices $v \in V$ ($\mathrm{O}(\log n)$ depth and $\mathrm{O}(|x|)$ work).

(2) Find the element with the smallest node index $v$ w.r.t. the random node order whose corresponding leaf is not marked as discarded ($\mathrm{O}(\log n)$ depth and $\mathrm{O}(|x|)$ work). Set $r(x)_v \leftarrow x_v$.

(3) Mark each leaf $w$ for which $x_v \le x_w$, including $v$, as discarded ($\mathrm{O}(1)$ depth and $\mathrm{O}(|x|)$ work).

(4) If there are non-discarded leaves ($\mathrm{O}(\log n)$ depth and $\mathrm{O}(|x|)$ work), continue at step (2).

Note that for each $w \neq v$ for which the corresponding node is discarded, we have $r(x)_w = \infty$. On the other hand, by construction we have for all $v$ for which we stored $r(x)_v = x_v$ that there is no $w \in V$ satisfying both $x_w \le x_v$ and $w < v$. Thus, the computed list is indeed $r(x)$.

The depth and work bounds follow from the complexity of the individual steps stated above and by observing that in each iteration, we add a distinct vertex–distance pair with non-$\infty$ value to the list that after termination equals $r(x)$. □

Based on Lemmas 17.8 and 17.9, Lemma 17.10 establishes that w.h.p. each of the intermediate results required by the oracle can be computed efficiently. The challenge is that the use of Lemma 17.8, which guarantees that the LE lists are short w.h.p., requires the lists and the random node order to be independent. As intermediate computations by the oracle make heavy use of intermediate filtering — compare, e.g., Equation (15.31) — the intermediate results do depend on the node order. The key insight, however, is that due to the properties of MBF-like algorithms, leaving out the intermediate filtering steps does not change the result of the computation by Corollary 13.20. We leverage this to remove said dependence, allowing us to apply Lemma 17.8.

Any intermediate result used by the oracle is of the form $r^V A_\lambda y$ with

$$y = (r^V A_\lambda)^f P_\lambda x^{(h)}, \tag{17.37}$$

where $x^{(h)} = r^V A_H^h x^{(0)}$ is the intermediate result of $h$ iterations on $H$, $\lambda \in \{0, \dots, \Lambda\}$ is a level, and $(r^V A_\lambda)^f P_\lambda$ represents another $f$ iterations in $G$ with edge weights stretched according to level $\lambda$. The oracle uses this to simulate the $(h+1)$-th iteration in $H$, compare Chapter 15 and Theorem 15.7 in particular.

**Lemma 17.10.** *Consider $x^{(0)} \in \mathcal{D}^V$ from Equation* (17.14). *For arbitrary $h, f, \lambda \in \mathbb{N}_0$, we can w.h.p.*

*(1) determine $r^V x^{(0)}$ from $x^{(0)}$ using $\mathrm{O}(n)$ work and $\mathrm{O}(1)$ depth,*

*(2) compute $r^V A_\lambda y$ from $y$ as defined in Equation* (17.37) *using $W \in \mathrm{O}(m \log^2 n)$ work and $D \in \mathrm{O}(\log^2 n)$ depth, and*

*(3) compute $r^V(\bigoplus_{\lambda=0}^{\Lambda} y_\lambda)$ from the individual $y_\lambda = (P_\lambda r^V A_\lambda^d P_\lambda)x^{(i)}$, $0 \le \lambda \le \Lambda$, with $W_\oplus \in \mathrm{O}(n \log^4 n)$ work and $D_\oplus \in \mathrm{O}(\log^2 n)$ depth.*

*Proof.* We establish the claims in order.

(1) Regarding the first claim, observe that $r^V x^{(0)} = x^{(0)}$. Hence, we can copy $x^{(0)}$ using $\mathrm{O}(\sum_{v \in V} |x_v^{(0)}|) = \mathrm{O}(n)$ work and constant depth.

(2) As for the second claim, we expand $x^{(h)}$ in Equation (17.37) and remove all intermediate filtering steps, obtaining

$$y \overset{(15.31)}{=} (r^V A_\lambda)^f P_\lambda \left( r^V \left( \bigoplus_{\lambda=0}^{\Lambda} P_\lambda (r^V A_\lambda)^d P_\lambda \right) \right)^h r^V x^{(0)} \tag{17.38}$$

$$\overset{(13.39)}{=} r^V \underbrace{A_\lambda^f P_\lambda \left( \bigoplus_{\lambda=0}^{\Lambda} P_\lambda A_\lambda^d P_\lambda \right)^h x^{(0)}}_{=:y'} . \tag{17.39}$$

The key observation is that — since the random order of $V$ only plays a role for $r$ and we removed all intermediate applications of $r^V$ — $y'$ does not depend on that order. Hence, we may apply Lemma 17.8 which yields that for each $v \in V$, $|y_v| = |r(y'_v)| \in \mathrm{O}(\log n)$ w.h.p. Condition on $|y_v| \in \mathrm{O}(\log n)$ for all $v \in V$ in the following, which happens w.h.p. by Lemma 2.7.

Regarding the computation of $r^V A_\lambda y$, we first compute each $(A_\lambda y)_v$ in parallel for all $v \in V$. By Lemma 13.3 and because $|y_v| \in \mathrm{O}(\log n)$, this can be done using $\mathrm{O}(\log n)$ depth and work

$$\mathrm{O}\left(\sum_{v\in V}\sum_{\substack{w\in V\\ \{v,w\}\in E}} |y_w| \log n\right) \subseteq \mathrm{O}\left(\sum_{\{v,w\}\in E} \log^2 n\right) = \mathrm{O}\left(m \log^2 n\right). \tag{17.40}$$

Here we use that propagation w.r.t. $\mathcal{D}$ — uniformly increasing weights — requires, due to $|y_v| \in \mathrm{O}(\log n)$, no more than $\mathrm{O}(1)$ depth and $\mathrm{O}(m \log n)$ work and is thus dominated by aggregation. To bound the cost of computing $r^V A_\lambda y$ from $A_\lambda y$, observe that we have

$$|(A_\lambda y)_v| \in \mathrm{O}\left(\sum_{\substack{w\in V\\ \{v,w\}\in E}} |y_w|\right). \tag{17.41}$$

Hence, by Lemma 17.9 and due to $|y_v| \in \mathrm{O}(\log n)$, we can compute $(r^V A_\lambda y)_v$ in parallel for all $v \in V$ using $\mathrm{O}(\log^2 n)$ depth and

$$\begin{aligned}
\mathrm{O}\left(\sum_{v\in V} |(A_\lambda y)_v| \log n\right) &\overset{(17.41)}{\subseteq} \mathrm{O}\left(\sum_{v\in V}\sum_{\substack{w\in V\\ \{v,w\}\in E}} |y_w| \log n\right) && (17.42)\\
&\subseteq \mathrm{O}\left(\sum_{\{v,w\}\in E} \log^2 n\right) && (17.43)\\
&\subseteq \mathrm{O}\left(m \log^2 n\right) && (17.44)
\end{aligned}$$

work. All operations are possible using $D \in \mathrm{O}(\log^2 n)$ depth and $W \in \mathrm{O}(m \log^2 n)$ work. As we condition only on an event that occurs w.h.p., this concludes the proof of the second claim.

(3) Regarding the last claim, condition on logarithmic length of all LE lists, i.e., on $|y_{\lambda v}| \in \mathrm{O}(\log n)$ for all $0 \leq \lambda \leq \Lambda$. We can compute $\bigoplus_{\lambda=0}^{\Lambda} y_{\lambda v} \in \mathcal{D}$, the aggregation for a single vertex $v$, using $\mathrm{O}(\sum_{\lambda=0}^{\Lambda} |y_{\lambda v}| \log n) = \mathrm{O}(\log^3 n)$ work and $\mathrm{O}(\log n)$ depth by Lemma 13.3. As the work bounds the length of the resulting list, we can determine $r(\bigoplus_{\lambda=0}^{\Lambda} y_{\lambda v})$ using $\mathrm{O}(\log^4 n)$ work and $\mathrm{O}(\log^2 n)$ depth by Lemma 17.9. Doing this in parallel for all $v \in V$ yields $W_\oplus \in \mathrm{O}(n \log^4 n)$ work and $D_\oplus \in \mathrm{O}(\log^2 n)$ depth. As we condition on two events that occur w.h.p., this concludes the last claim.

Together, the claims are established. □

## 17.4 Efficient Parallel Metric Tree Embeddings

Determining the LE lists of $H$ yields a probabilistic tree embedding of $G$ with expected stretch $\mathrm{O}(\log n)$ (Section 17.1), is the result of an MBF-like algorithm (Section 17.2), and each iteration of this algorithm is efficient (Theorem 15.7 and Section 17.3). We assemble these pieces in Theorem 17.11, which relies on $G$ containing a suitable hop set; Corollaries 17.12 and 17.13 remove this assumption by first invoking known algorithms to establish this property. Note that Theorem 17.11 serves as a blueprint yielding improved tree embedding algorithms when provided with improved hop-set algorithms.

**Theorem 17.11.** *Suppose we know the weighted incidence list of a graph $G = (V, E, \omega)$ that contains a $(d, \hat{\varepsilon})$-hop set, i.e., that satisfies $\operatorname{dist}^d(v, w, G) \leq (1 + \hat{\varepsilon}) \operatorname{dist}(v, w, G)$ for all $v, w \in V$, where $d \in \mathbb{N}$ and $\hat{\varepsilon} \in \mathbb{R}_{>0}$. Then we can w.h.p. sample a tree embedding of $G$ of expected stretch $\mathrm{O}((1+\hat{\varepsilon})^{\mathrm{O}(\log n)} \log n)$ using depth $\mathrm{O}(d \log^4 n) \subseteq \tilde{\mathrm{O}}(d)$ and work $\mathrm{O}(m(d + \log n) \log^5 n) \subseteq \tilde{\mathrm{O}}(md)$.*

*Proof.* We first w.h.p. compute the LE lists of $H$, see Definition 15.2 and Theorem 15.5. To this end, by Lemma 17.10, we may apply Theorem 15.7 with parameters $D \in \mathrm{O}(\log^2 n)$, $W \in \mathrm{O}(m \log^2 n)$, $D_\oplus \in \mathrm{O}(\log^2 n)$, and $W_\oplus \in \mathrm{O}(n \log^4 n)$; we arrive at

$$\mathrm{O}\left((dD + D_\oplus) \log^2 n\right) = \mathrm{O}\left(d \log^4 n\right) \tag{17.45}$$

depth and

$$\begin{aligned} \mathrm{O}\left((dW \log n + W_\oplus) \log^2 n\right) &= \mathrm{O}\left((dm + n \log n) \log^5 n\right) && (17.46)\\ &\subseteq \mathrm{O}\left(m(d + \log n) \log^5 n\right) && (17.47) \end{aligned}$$

work. Note that the bounds of $D$, $W$, $D_\oplus$, and $W_\oplus$ hold w.h.p. for a single iteration. As there is only a polynomially bounded number of iterations, however, the stated bounds w.h.p. hold for all of them by Lemma 2.7.

In order to compute the according FRT tree $T$, observe that the LE lists of $H$ have length $\mathrm{O}(\log n)$ w.h.p. by Lemma 17.8. Hence, we can compute $T$ using depth $\mathrm{O}(\log^2 n)$ and work $\mathrm{O}(n \log^3 n)$ by Lemma 17.4. The depth and work of this step are dominated by the above-stated depth and work for invoking Theorem 15.7.

It remains to show that $T$ is a tree embedding, compare Definition 17.1, of $G$ with the expected stretch stated above. As shown by Fakcharoenphol et al. [50], $T$ is a tree embedding of $H$ with an expected stretch of $\mathrm{O}(\log n)$ and by Theorem 15.5, w.h.p. $\operatorname{dist}(v, w, G) \leq \operatorname{dist}(v, w, H) \leq (1 + \hat{\varepsilon})^{\mathrm{O}(\log n)} \operatorname{dist}(v, w, G)$. Hence, we obtain

$$\operatorname{dist}(v, w, G) \overset{(15.16)}{\leq} \operatorname{dist}(v, w, H) \overset{[50]}{\leq} \operatorname{dist}(v, w, T) \tag{17.48}$$

and, in expectation,

$$\begin{aligned} \operatorname{dist}(v, w, T) &\overset{[50]}{\in} \mathrm{O}(\operatorname{dist}(v, w, H) \log n) && (17.49)\\ &\overset{(15.16)}{\subseteq} \mathrm{O}\left((1 + \hat{\varepsilon})^{\mathrm{O}(\log n)} \operatorname{dist}(v, w, G) \log n\right), && (17.50) \end{aligned}$$

concluding the proof. □

As stated above, we require $G$ to contain a $(d, 1/\operatorname{polylog} n)$-hop set with $d \in \operatorname{polylog} n$ in order to achieve polylogarithmic depth. We also need to determine such a hop set using $\operatorname{polylog} n$ depth and near-linear work in $m$ and that it does not significantly increase the problem size by adding too many edges. With Cohen's hop set [34] we can meet these requirements — up to the extra factor of $m^{\varepsilon}$ in work — resulting in Corollary 17.12. Observe that our work almost matches the sequential state of the art, the algorithm of Blelloch et al. that requires $\mathrm{O}(m \log n)$ time w.h.p. [20].

**Corollary 17.12.** *Given a weighted incidence list of a graph $G$ and a constant $0 < \varepsilon \leq \frac{1}{2}$, we can w.h.p. sample a tree embedding of $G$ of expected stretch $\mathrm{O}(\log n)$ using depth* $\operatorname{polylog} n$ *and work $\tilde{\mathrm{O}}(m^{1+\varepsilon})$.*

*Proof.* We apply the hop-set construction by Cohen [34] to $G = (V, E, \omega)$ to w.h.p. determine an intermediate graph $G'$ with vertices $V$ and an additional $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ edges. The algorithm guarantees $\operatorname{dist}^{d}(v, w, G) \leq (1 + \hat{\varepsilon}) \operatorname{dist}(v, w, G')$ for $d, \hat{\varepsilon}^{-1} \in \operatorname{polylog} n$, where the exponent of the $\operatorname{polylog} n$ term for $\hat{\varepsilon}$ is under our control, and has depth $\operatorname{polylog} n$ and work $\tilde{\mathrm{O}}(m^{1+\varepsilon})$. Choosing $\hat{\varepsilon} \in \mathrm{O}(1/\log^2 n)$ and applying Theorem 17.11, the claim follows due to Equation (15.19). □

Adding a hop set to $G$, embedding the resulting graph in $H$, and sampling an FRT tree on $H$ are three nested embeddings of $G$. Still, in terms of stretch, the embedding of Corollary 17.12 is — up to a factor in $1 + \mathrm{o}(1)$ — as good as directly constructing an FRT tree of $G$: (1) Hop sets do not stretch distances. (2) By Theorem 15.5 and Equation (15.19), $H$ introduces a stretch of $1 + 1/\operatorname{polylog} n$. (3) Together, this ensures that the expected stretch of the FRT embedding w.r.t. $G$ is $\mathrm{O}(\log n)$.

It is possible to reduce the work at the expense of an increased stretch by first applying the spanner construction by Baswana and Sen [10]. Refer to Chapter 16 for a brief discussion of its most relevant properties w.r.t. our model of computation and our applications. Essentially, an additional factor of $\mathrm{O}(\varepsilon^{-1})$ in the stretch allows us to reduce the work from $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ to $\tilde{\mathrm{O}}(m + n^{1+\varepsilon})$. Observe that at this point, one can take the position that $\varepsilon$ is a constant and that, accordingly, Corollary 17.13 is stronger than Corollary 17.12 as it provides the same asymptotic guarantees with lower work. On the other hand, one may argue that a constant overhead in the stretch does matter as, after all, the expected stretch from Corollary 17.12 is essentially — up to a factor in $1 + \mathrm{o}(1)$ — exactly the same as that of an FRT embedding of $G$.

**Corollary 17.13.** *Suppose we are given the weighted incidence list of a graph $G$ and a constant $\varepsilon \in \mathbb{R}$ with $0 < \varepsilon \leq \frac{5}{4}$. Then we can w.h.p. compute a tree embedding of $G$ with an expected stretch of $(2\lceil \varepsilon^{-1} \rceil - 1)\alpha \in \mathrm{O}(\varepsilon^{-1} \log n) = \mathrm{O}(\log n)$, where $\alpha \in \mathrm{O}(\log n)$ is the expected stretch of Corollary 17.12, using depth* $\operatorname{polylog} n$ *and work $\tilde{\mathrm{O}}(m + n^{1+\varepsilon})$.*

*Proof.* Choose $\varepsilon' := \sqrt{1 + \varepsilon} - 1$ and $k := \lceil \varepsilon'^{-1} \rceil$; observe that $k \in \mathbb{N}$ and $0 < \varepsilon' \leq \frac{1}{2}$. We determine a $(2k - 1)$-spanner $G_S$ of $G$ and apply Corollary 17.12 to $G_S$. The detailed procedure is the following.

(1) As detailed in Chapter 16, the algorithm of Baswana and Sen [10] computes a $(2k-1)$-spanner of $G = (V, E, \omega)$, i.e., a subgraph $G_S = (V, E_S, \omega)$ satisfying for all $v, w \in V$ that $\operatorname{dist}(v, w, G) \leq \operatorname{dist}(v, w, G_S) \leq (2k - 1) \operatorname{dist}(v, w, G)$. This requires $\tilde{\mathrm{O}}(k)$ depth and $\tilde{\mathrm{O}}(km)$ work, we obtain $|E_S| \in \tilde{\mathrm{O}}(kn^{1+1/k})$ w.h.p., and $k \in \mathrm{O}(\log n)$ w.l.o.g. by Observation 16.3.

The bound of $k \in \mathrm{O}(\log n)$ simplifies the depth to $\operatorname{polylog} n$, the work to $\tilde{\mathrm{O}}(m)$, and, w.h.p., the number of edges in the spanner to $|E_S| \in \tilde{\mathrm{O}}(n^{1+1/k})$.

(2) Apply Corollary 17.12 to $G_S$ and $\varepsilon'$, where the latter is within the permitted bounds as argued above. This step requires an additional polylog $n$ depth and

$$\tilde{\mathrm{O}}\left(|E_S|^{1+\varepsilon'}\right) = \tilde{\mathrm{O}}\left((n^{1+1/k})^{1+\varepsilon'}\right) \subseteq \tilde{\mathrm{O}}\left(n^{(1+\varepsilon')^2}\right) = \tilde{\mathrm{O}}\left(n^{1+\varepsilon}\right) \tag{17.51}$$

work, where the bound on work holds w.h.p.

Sequentially executing both steps yields polylog $n$ depth and w.h.p. $\tilde{\mathrm{O}}(m + n^{1+\varepsilon})$ work, as claimed. Let $\alpha$ be the expected stretch from Corollary 17.12, where $\alpha \in \mathrm{O}(\log n)$ w.h.p. As we only require constantly many events that occur w.h.p. to succeed, observing that the spanner increases the expected stretch to $(2k-1)\alpha = (2\lceil\varepsilon^{-1}\rceil - 1)\alpha \in \mathrm{O}(\varepsilon^{-1}\log n)$ concludes the proof. $\square$

## 17.5 Reconstructing Paths from Virtual Edges

Given that we deal with distances and not with paths in the construction of FRT trees, one concern needs to be addressed: Consider a graph $G = (V, E, \omega)$, its augmentation with a $(d, \hat{\varepsilon})$-hop set resulting in $G'$, which is then embedded into the complete graph $H = (V, \binom{V}{2}, \omega_H)$, and finally into an FRT tree $T = (V_T, E_T, \omega_T)$. How can an edge $e \in E_T$ of weight $\omega_T(e)$ be mapped to a path $p$ in $G$ with $\omega(p) \leq \omega_T(H)$? Note that it is desirable to answer this question in polylogarithmic depth and without incurring too much memory overhead. In Observations 17.14–17.16, we outline a three-step approach that maps edges of $T$ to paths in $H$, edges of $H$ to paths in $G'$, and finally edges of $G'$ to paths in $G$. Our purpose is not to provide specifically tailored data structures, but rather to informally sketch how this can be achieved within polylogarithmic depth.

**Observation 17.14.** *Concerning a tree edge $e \in E_T$, we claim that $e$ maps back to a path $p$ of at most* $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ *hops in $H$ with $\omega_H(p) \leq \omega_T(e)$.*

Identify each tree node $(v_i, \dots, v_k)$ — recall Section 17.1 and Definition 17.2 — with its *leading* node $v_i \in V$. In particular, each leaf $(v_0, \dots, v_k)$ is identified with the node in $v_0 = v \in V$. First consider the situation that $e = \{(v_0, v_1, \dots, v_k), (v_1, \dots, v_k)\} \in E_T$ is incident to a leaf of $T$ identified with $v_0$. In this case, the LE list of $v_0$ contains an entry $(v_1, \mathrm{dist}(v_0, v_1, H))$ that must fulfill $\mathrm{dist}(v_0, v_1, H) \leq 2^0\beta\,\omega_{\min}$ and we can trace the shortest $v_0$-$v_1$-path $p$ in $H$ based on the LE lists (nodes locally store the predecessor of shortest paths just like in APSP). By construction, $p$ fulfills

$$\omega_H(p) \leq 2^0\beta\,\omega_{\min} \leq 2^1\beta\,\omega_{\min} = \omega_T(e) \tag{17.52}$$

and has at most $\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ hops by Theorem 15.5.

Next, consider the case that $e = \{(v_i, v_{i+1}, \dots, v_k), (v_{i+1}, \dots, v_k)\}$ is not incident to a leaf of $T$. Choose an arbitrary leaf $v_0$ that is a descendant of $(v_i, v_{i+1}, \dots, v_k)$ (this choice can, e.g., be fixed when constructing the tree from the LE list without increasing the asymptotic bounds on depth or work). We can trace shortest paths from $v_0$ to $v_i$ and from $v_0$ to $v_{i+1}$ in $H$, respectively. The cost of their concatenation $p$, a path from

$v_i$ to $v_0$ to $v_{i+1}$, is

$$\begin{aligned}\omega_H(p) &= \operatorname{dist}(v_0, v_i, H) + \operatorname{dist}(v_0, v_{i+1}, H) &(17.53)\\ &\leq 2^{i-1}\beta\,\omega_{\min} + 2^i\beta\,\omega_{\min} &(17.54)\\ &< 2^{i+1}\beta\,\omega_{\min} &(17.55)\\ &= \omega_T(e) &(17.56)\end{aligned}$$

by the properties of FRT trees. Note that, due to the identification of each tree node with its leading graph node, paths in $T$ map to concatenable paths in $H$. Furthermore, $p$ has at most $2\,\mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ hops.

**Observation 17.15.** *We can map an edge $e = \{v, w\}$ of $H$ to a $d$-hop shortest $v$-$w$-path $p$ in $G'$ using $\tilde{\mathrm{O}}(nd)$ memory. The mapping guarantees $\omega(p) \leq \omega_H(e)$, where $\omega$ is appropriately extended to account for the hop-set edges in $G'$.*

Regarding the mapping from an edge $e$ of $H$ to a path in $G'$, recall that we compute the LE lists of $H$ by repeated application of the operations $r^V$, $\oplus$, $P_\lambda$, and $A_\lambda$ with $0 \leq \lambda \leq \Lambda$, compare Chapter 15. Observe that $r^V$, $\oplus$, and $P_\lambda$ discard information, i.e., distances to nodes that do not make it into the final LE lists and therefore are irrelevant to routing. $A_\lambda$, on the other hand, is an MBF-like iteration. Thus, we may store the necessary information for tracing back the induced paths at each node. Specifically, we can store, for each iteration $h \leq \mathrm{SPD}(H) \in \mathrm{O}(\log^2 n)$ w.r.t. $H$, each of the intermediate $d$ iterations in $G'$, and each $\lambda \leq \Lambda \in \mathrm{O}(\log n)$, the state vector $y$ of the form in Equation (17.37) in a lookup table. This requires $\tilde{\mathrm{O}}(d)$ memory per node and maps edges of $H$ to $d$-hop shortest paths in $G'$. The guarantee that $\omega(p) \leq \omega_H(e)$ follows from observing that $p$ is a $d$-hop shortest $v$-$w$-path in $G'$ and that we have $\omega(p) = \operatorname{dist}^d(v, w, G') \leq \omega_H(e)$ by Definition 15.2.

**Observation 17.16.** *Mapping edges of $G'$ to paths in $G$ depends on the hop set. This mapping does not increase the weight.*

Cohen [34] does not discuss this in her article, but her hop-set edges can be efficiently mapped to paths in the original graph by a lookup table: Hop-set edges either correspond to a shortest path in a small cluster, or to a path that has been explored using polylogarithmic depth. Regarding other hop-set algorithms, we note that many techniques constructing hop-set edges using depth $D$ allow reconstructing corresponding paths at depth $\mathrm{O}(D)$, i.e., that polylogarithmic-depth algorithms are compatible analogously to Cohen's hop sets. For instance, this is the case for the hop-set construction by Henzinger et al. [73] which we leverage in Chapter 18. This observation, however, needs to be carefully checked for each hop-set construction. The fact that path weights do not increase follows from the requirement that hop sets do not decrease distances; a hop-set edge $\{v, w\}$ must have a weight of at least $\operatorname{dist}(v, w, G)$ [34, 73].

## 17.6 Tree Embeddings without Steiner Nodes

As detailed by Blelloch et al. [21], it is possible to obtain from an FRT tree a metric tree embedding that contains no Steiner nodes while maintaining the expected asymptotic stretch. To this end, label each node of the original FRT tree with the smallest node in its subtree and contract all edges between nodes with the same label. Non-contracted edges can be mapped to paths in $H$ just like in Observation 17.14 when using the label — the smallest leaf — instead of an arbitrary leaf. A formal argument on why this yields the same asymptotic stretch as the original FRT tree is given in Lemma 3.12 of Blelloch et al. [21]. Note that Blelloch et al. explicitly stretch all edges of the original FRT tree by a factor of 2, while we already account for this factor in Definition 17.2. Observe that the labels uniquely determine the tree and can be determined using $\mathrm{O}(n \log n)$ work and $\mathrm{O}(\log n)$ depth by propagating the minima up the original FRT tree.

**Observation 17.17.** *Corollaries 17.12 and 17.13 still hold when not permitting the tree embedding to use Steiner nodes. Observation 17.14 applies to the resulting trees.*

# CHAPTER 18

# Distributed FRT Embeddings

In this chapter, we show that distributed algorithms for constructing FRT-type tree embeddings in the CONGEST model are covered by our framework as well. We improve upon the state of the art by reducing the round complexity of determining a tree embedding of expected stretch $\mathrm{O}(\log n)$ in the CONGEST model.

We introduce the CONGEST model in Section 18.1. In Sections 18.2 and 18.3, we briefly summarize the algorithms by Khan et al. [81] and by Ghaffari and Lenzen [65]. Our framework permits doing this in a compact way. We use these preliminaries, our machinery, and a distributed hop-set construction due to Henzinger et al. [73] in Section 18.4, where we propose an algorithm that reduces a multiplicative overhead of $n^{\varepsilon}$ in the round complexity of Ghaffari and Lenzen [65] to $n^{\mathrm{o}(1)}$. Note that replacing the hop set is straightforward since our theorems in the previous chapters are formulated w.r.t. generic $(d, \hat{\varepsilon})$-hop sets. Furthermore, our algorithm improves the expected stretch: We replace the multiplicative penalty of $\mathrm{O}(\varepsilon^{-1})$ by Ghaffari and Lenzen [65] in the expected stretch by $1 + \mathrm{o}(1)$.

Throughout this chapter, let $G = (V, E, \omega)$ be a weighted graph and denote, for any graph $G$, by $A_G \in (\mathbb{R}_{\geq 0} \cup \{\infty\})^{V \times V}$ its adjacency matrix according to Equation (12.17). Fix the semiring $\mathcal{S} = \mathcal{S}_{\min,+}$, the zero-preserving semimodule $\mathcal{M} = \mathcal{D}$ from Definition 13.1, and $r$, $\sim$ and $x^{(0)}$ as given in Definition 17.5.

## 18.1 The CONGEST Model

The CONGEST model captures distributed computations performed by the nodes of a graph — in this context commonly referred to as *network* — where communication is restricted to the graph's edges. Furthermore, the volume of communication per edge and time unit is limited, hence the name of the model. We refer to Lynch [107] and Peleg [119] for formal definitions of distributed models of computation. In this section, we briefly outline the core aspects of the CONGEST model.

Each node is initialized with a unique *identifier (ID)* of $\mathrm{O}(\log n)$ bits, knows the IDs of its adjacent nodes along with the weights of the corresponding incident edges, and its part of the input. The objective is for each node to compute its part of the output. Regarding the computation of FRT trees, the input of a node is empty and its output, as detailed in Chapter 17, is its LE list.[1]

Computations are organized in perfectly synchronous *rounds*. In each round, each node does the following:

[1]In the CONGEST model, it is not feasible for each node determine the entire FRT tree in $\mathrm{o}(n)$ rounds: This requires communicating at least the random node order to each node, taking $\Omega(n)$ rounds of communication in the presence of nodes of degree one.

(1) Perform finite but otherwise arbitrary[2] local computations.

(2) Send a, possibly empty, message of $\mathrm{O}(\log n)$ bits to each neighboring node. It is not required to send the same message to all neighbors.

(3) Receive the messages sent by the neighbors in the current round, where the sending node of a message is known.

We are interested in the *round complexity,* i.e., in how many rounds it takes for an algorithm to complete. Recall that edge weights can be encoded using $\mathrm{O}(\log n)$ bits by assumption. Hence, we may encode constantly many ID–distance pairs in a single message.

**LOCAL** Observe that without congestion — this is referred to as the LOCAL model in which we drop the restriction regarding message size — every problem can be solved in $\mathrm{D}(G)$ rounds [119]: Every node broadcasts its ID and weighted adjacency list, all nodes acquire full information about the network, and each node locally computes its part of the output. Further observe that $\Omega(\mathrm{D}(G))$ rounds also are a lower bound for determining an FRT tree in the LOCAL model: From the resulting FRT tree, every node can derive a 2-approximation of its distance to the vertex with the smallest ID — the "root of the FRT tree" — in a weighted line graph, which is not possible without learning about edge weights that are $\Omega(\mathrm{D}(G))$ hops away from the node at one end of the line.

**Bounds** In the presence of congestion, however, there are stronger lower bounds for many problems [38]. A prominent example is the lower bound of $\Omega(\sqrt{n}/\log n + \mathrm{D}(G))$ on the round complexity of determining an MST. It holds for deterministic, exact MST computations [120], as well as for randomized and approximate MST algorithms [38, 47]. Regarding the FRT embedding, Ghaffari and Lenzen lower-bound the round complexity of determining any tree embedding by $\tilde{\Omega}(\sqrt{n} + \mathrm{D}(G))$, assuming that it has non-trivial expected stretch [65].

**Termination** Regarding the termination of a distributed algorithm, we distinguish between *local termination* and *global termination.* Local termination means that a node stops sending messages and global termination means that all nodes learn that all local computations are finished, i.e., that the algorithm reached a fixed point. We are interested in global termination, as constructing a metric tree embedding is an inherently global problem.

As all algorithms presented in the chapter require $\Omega(\mathrm{D}(G))$ rounds of computation, we can afford to detect global termination using standard techniques. With a BFS tree in place, each of its leafs in each round sends a bit to its parent that indicates whether it considers the computation locally complete. Non-leaf nodes send the logical And of the bits received from their children and their own opinion. The root of the BFS tree can decide whether the computation is globally over and initiate an appropriate broadcast once that happens. All of the above steps, including the construction of a BFS tree, are possible with an additive overhead of $\mathrm{O}(\mathrm{D}(G))$ in the round complexity.

[2]Some authors restrict the time for local computations [119]. Polynomial time, however, suffices for all algorithms in this chapter.

## 18.2 The Algorithm of Khan et al.

In our terminology, the algorithm of Khan et al. [81] performs $\mathrm{SPD}(G)$ iterations of the MBF-like algorithm for collecting LE lists implied by Definition 17.5, compare Chapter 17. This means that the algorithm determines

$$r^V A_G^{\mathrm{SPD}(G)} x^{(0)} \overset{(13.39)}{=} \left(r^V A_G\right)^{\mathrm{SPD}(G)} x^{(0)}, \tag{18.1}$$

where each node $v \in V$ learns its own LE list $(r^V x^{(\mathrm{SPD}(G))})_v$. The algorithm works in $\mathrm{SPD}(G)+1$ iterations by initializing $x^{(0)}$ as in Equation (17.14) and iteratively computing $x^{(i+1)} = r^V A_G x^{(i)}$ until a fixed point is reached, i.e., until $x^{(i+1)} = x^{(i)}$.

As $(r^V A_G)^i x^{(0)} = r^V A_G^i x^{(0)}$, Lemma 17.8 shows that w.h.p. $|x_v^{(i)}| \in \mathrm{O}(\log n)$ for all iterations $i$ and all $v \in V$. Therefore, each node $v \in V$ can w.h.p. transmit its LE list $x_v^{(i)}$ to all of its neighbors entry by entry using $\mathrm{O}(\log n)$ rounds. Upon reception of its neighbors' lists, $v$ locally computes $x_v^{(i+1)}$. Thus, each iteration takes $\mathrm{O}(\log n)$ rounds w.h.p., implying the round complexity of $\mathrm{O}(\mathrm{SPD}(G) \log n)$ w.h.p. shown by Khan et al. [81].

## 18.3 The Algorithm of Ghaffari and Lenzen

The strongest lower bound regarding the round complexity for constructing a low-stretch metric tree embedding of $G$ in the CONGEST model is $\tilde{\Omega}(n^{1/2} + \mathrm{D}(G))$, see Section 18.1. In the case that $\mathrm{SPD}(G) \gg \max\{\mathrm{D}(G), n^{1/2}\}$ one may thus hope for a solution that runs in $\tilde{o}(\mathrm{SPD}(G))$ rounds and, indeed, Ghaffari and Lenzen show for any $\varepsilon \in \mathbb{R}_{>0}$ that expected stretch $\mathrm{O}(\varepsilon^{-1} \log n)$ can be achieved in $\tilde{\mathrm{O}}(n^{1/2+\varepsilon} + \mathrm{D}(G))$ rounds [65]. In the following, we summarize their approach.

The strategy is to first determine the LE lists of a constant-stretch metric embedding of (the induced submetric of) an appropriately sampled subset of $V$. The resulting graph is called the skeleton spanner, and its LE lists are used to jump-start the computation in the remaining graph. When sampling the skeleton nodes in the right way, stretching non-skeleton edges analogously to Chapter 15, and fixing a shortest path for each pair of vertices, w.h.p. all of these paths contain a skeleton node within a few hops. Ordering skeleton nodes before non-skeleton nodes w.r.t. the random order implies that each LE list has a short prefix accounting for the local neighborhood, followed by a short suffix containing skeleton nodes only. This is due to the fact that skeleton nodes dominate all non-skeleton nodes for which the respective shortest path passes through them. Hence, no node has to learn information that is further away than $d_S$, an upper bound on the number of hops when a skeleton node is encountered on a shortest path that holds w.h.p.

**The Graph $H$** Ghaffari and Lenzen embed $G = (V, E, \omega)$ into $H$ and sample an FRT tree on $H$, where $H$ is derived as follows. Sample $S \subseteq V$, where each $v \in V$ joins $S$ independently with probability $1/\sqrt{n}$. For a sufficiently large constant $c \in \mathbb{R}_{\geq 1}$, abbreviate $\ell := \lceil c\sqrt{n}\log n \rceil$. Define the *skeleton graph* as

$$\begin{aligned} G_S &:= (S, E_S, \omega_S)\,, \text{ where} && (18.2)\\ E_S &:= \left\{ \{s,t\} \in \binom{S}{2} \mid \operatorname{dist}^\ell(s,t,G) < \infty \right\} \text{ and} && (18.3)\\ \omega_S(s,t) &:= \operatorname{dist}^\ell(s,t,G). && (18.4) \end{aligned}$$

Then w.h.p. $\operatorname{dist}(s,t,G_S) = \operatorname{dist}(s,t,G)$ for all $s,t \in S$ (Lemma 10 of Ghaffari and Lenzen [65]). Unfortunately, it is not possible to determine $G_S$ exactly in $\mathrm{o}(n)$ rounds [103]; hence, $G_S$ is not explicitly constructed.

Instead, however, it is possible to determine a spanner of $G_S$ using the construction of Baswana and Sen [10] and to make it global knowledge. For $k \in \Theta(\varepsilon^{-1})$, construct a $(2k-1)$-spanner

$$G'_S := \left(S, E'_S, \omega_S\right) \tag{18.5}$$

of the skeleton graph $G_S$ that has $\tilde{\mathrm{O}}(kn^{1/2+1/k}) \subseteq \tilde{\mathrm{O}}(n^{1/2+\varepsilon})$ edges w.h.p. (compare Chapter 16 and Theorem 11 of Ghaffari and Lenzen [65]). Define

$$\begin{aligned} H &:= (V, E_H, \omega_H), \text{ where} && (18.6)\\ E_H &:= E \cup E'_S, \text{ and} && (18.7)\\ \omega_H(e) &:= \begin{cases} \omega_S(e) & \text{if } e \in E'_S \text{ and}\\ (2k-1)\,\omega(e) & \text{otherwise.} \end{cases} && (18.8) \end{aligned}$$

By construction, $G$ embeds into $H$ with a stretch of $2k-1$ w.h.p., i.e.,

$$\operatorname{dist}(v,w,G) \le \operatorname{dist}(v,w,H) \le (2k-1)\operatorname{dist}(v,w,G). \tag{18.9}$$

Computing an FRT tree $T$ of $H$ of expected stretch $\mathrm{O}(\log n)$ thus implies that $G$ embeds into $T$ with expected stretch $\mathrm{O}(k \log n) = \mathrm{O}(\varepsilon^{-1}\log n)$.

**FRT Trees of $H$** Observe that min-hop shortest paths in $H$ contain only a single maximal subpath consisting of spanner edges, where the maximal subpaths of non-spanner edges have at most $\ell$ hops w.h.p. This follows analogously to Lemma 15.4 with two levels and a sampling probability of $1/\sqrt{n}$.

Assume $s < v$ for all $s \in S$ and $v \in V \setminus S$; we establish this assumption below. With that assumption in mind, consider $s \in S$ and $v, w \in V \setminus S$. The only way an entry $(s, \operatorname{dist}(v,s,H))$ of the LE list of $v \in V$ can be dominated by $(w, \operatorname{dist}(v,w,H))$ is that $w$ is closer to $v$ than $s$. In fact, no non-skeleton node $w \in V \setminus S$ is part of the LE list of $v$, unless $\operatorname{dist}(v,w,H) < \min\{\operatorname{dist}(v,s,H) \mid s \in S\}$. Hence, the LE list of $v$ has some entries accounting for the local neighborhood and only $S$-entries with higher distances.

Furthermore, for each $v \in V$ and each entry $(w, \operatorname{dist}(v,w,H))$ of its LE list, there w.h.p. is a min-hop shortest $v$-$w$-path with a prefix of at most $\ell$ non-spanner edges followed by a shortest path in $G'_S$. This entails that w.h.p.

$$r^V A_H^{\mathrm{SPD}(H)} x^{(0)} = r^V A_{G,2k-1}^{\ell} A_{G'_S}^{|S|} x^{(0)} = r^V A_{G,2k-1}^{\ell} \underbrace{\left(r^V A_{G'_S}^{|S|} x^{(0)}\right)}_{=:\bar{x}^{(0)}}, \tag{18.10}$$

where $A_{G,s}$ is $A_G$ with entries stretched by factor of $s \in \mathbb{R}_{>0}$ and we extend $A_{G'_S}$ to a $V \times V$ matrix by setting, for all $v, w \in V \setminus S$, $(A_{G'_S})_{vw} = \infty$ if $v \neq w$, and $(A_{G'_S})_{vw} = 0$ otherwise (the equivalent of treating $V \setminus S$ as singleton vertices).

In order to construct an FRT tree, suppose we have sampled uniform permutations of $S$ and $V \setminus S$, and a random choice of $\beta$. We extend the permutations to a permutation of $V$ by ruling that for all $s \in S$ and $v \in V \setminus S$, we have $s < v$, fulfilling the above assumption. Lemma 20 of Ghaffari and Lenzen [65] shows that the introduced dependence between the topology of $H$ and the resulting permutation on $V$ does not increase the expected stretch of the embedding beyond $\mathrm{O}(\log n)$. The crucial advantage of this approach lies in the fact that the LE lists of nodes in $G'_S$, $\bar{x}^{(0)}$ in Equation (18.10), may be used to jump-start the construction of LE lists of $H$, i.e., the computation of $A^{\ell}_{G,2k-1}\bar{x}^{(0)}$ in Equation (18.10).

**The Algorithm** In the CONGEST model, the LE lists of $H$ can be determined in $\tilde{\mathrm{O}}(n^{1/2+\varepsilon} + \mathrm{D}(G))$ rounds as follows [65].

(1) Some node $v_0 \in V$ starts the computation by broadcasting a random choice of $\beta$, constructing a BFS tree on the fly. Upon receipt, each node generates a random ID of $\mathrm{O}(\log n)$ bits which is unique w.h.p. This takes $\mathrm{O}(\mathrm{D}(G))$ rounds. Finding the minimum and the maximum vertex ID as well as counting the amount of nodes with an ID of less than some threshold via the BFS tree takes $\mathrm{O}(\mathrm{D}(G))$ rounds. Hence, $v_0$ can determine the $\lceil \sqrt{n} \rceil$-th node ID via binary search in $\mathrm{O}(\mathrm{D}(G) \log n)$ rounds; this determines the set $S$ and satisfies the assumption used in Equation (18.10). Together, $\tilde{\mathrm{O}}(\mathrm{D}(G))$ rounds suffice.

(2) The nodes in $S$ determine $G'_S$, such that all $v \in V$ learn $E'_S$ and $\omega_S$. By Theorem 11 of Ghaffari and Lenzen [65], this is possible in $\tilde{\mathrm{O}}(n^{1/2+\varepsilon} + \mathrm{D}(G))$ rounds. This enables all $v \in V$ to locally determine $\bar{x}^{(0)}_v$, which is trivial for $v \notin S$ — in this case, the LE list only contains the entry $(v, 0)$ — and uniquely determined by the node IDs and $G'_S$ otherwise.

(3) Subsequently, all nodes w.h.p. determine, via $\ell \in \mathrm{O}(n^{1/2} \log n)$ MBF-like iterations of
$$\bar{x}^{(i+1)} := r^V A_{G,2k-1}\bar{x}^{(i)}, \tag{18.11}$$
their component of $\bar{x}^{(h)} = r^V A^{\ell}_{G,2k-1}\bar{x}^{(0)}$. Here, one exploits $|\bar{x}^{(i)}_v| \in \mathrm{O}(\log n)$ w.h.p. for all $0 \leq i \leq \ell$ by Lemma 17.8.[3] Furthermore, filtering is a local operation. Thus, each iteration can be performed by transmitting an LE list of $\mathrm{O}(\log n)$ entries over each edge, i.e., in $\mathrm{O}(\log n)$ rounds. The entire step hence requires $\tilde{\mathrm{O}}(\ell) \subseteq \tilde{\mathrm{O}}(n^{1/2})$ rounds.

Together, this w.h.p. implies the round complexity of $\tilde{\mathrm{O}}(n^{1/2+\varepsilon} + \mathrm{D}(G))$ for an embedding of expected stretch $\mathrm{O}(\varepsilon^{-1} \log n)$.

[3]We apply Lemma 17.8 twice, as it requires $x \in \mathcal{D}$ and the permutation to be independent. First consider a computation initialized with $y^{(0)}_{vw} := 0$ if $v = w \in S$ and $y^{(0)}_{vw} := \infty$ else. By Lemma 17.8, we have $|y^{(i)}_v| \in \mathrm{O}(\log n)$ w.h.p. for all $y^{(i)} := r^V A^i_{H_S} y^{(0)}$ and iterations $i \in \{1, \dots, |S|\}$. Analogously, apply Lemma 17.8 to $z^{(i)} := r^V A^i_{G,2k-1} z^{(0)}$, $i \in \{1, \dots, \ell\}$ with $z^{(0)}_{vw} := 0$ if $v = w \in V \setminus S$ and $z^{(0)}_{vw} := \infty$ else; this yields that $|z^{(i)}_v| \in \mathrm{O}(\log n)$ for all $v \in V$ w.h.p., too. As we have $x^{(i)}_v = r^V(y^{(j)}_v \oplus z^{(k)}_v)$ for all $v \in V$ and appropriate $i, j, k \in \mathbb{N}_0$, we obtain $|x^{(i)}_v| \in \mathrm{O}(\log n)$ w.h.p.

## 18.4 $\mathrm{O}(\log n)$ Stretch in Near-Optimal Time

The multiplicative overhead of $n^{\varepsilon}$ in the round complexity as well as the factor of $\mathrm{O}(\varepsilon^{-1})$ in stretch are due to constructing, broadcasting, and using the skeleton spanner. We improve upon this by relying on hop sets, just as we do in our parallel construction in Chapter 17. Henzinger et al. show how to compute an $(n^{\mathrm{o}(1)}, \mathrm{o}(1))$-hop set of the skeleton graph in the CONGEST model using $(n^{1/2} + \mathrm{D}(G))n^{\mathrm{o}(1)}$ rounds [73].

Our approach is similar to the one outlined in Section 18.3. The key difference is that we replace the skeleton spanner by an approximation of the skeleton graph which we augment with a hop set. We combine this with the techniques from Chapter 15. This permits the efficient construction of the LE lists of $S$ which can be used to jump-start the construction of LE lists for all nodes. However, we construct the LE lists of $S$ in a distributed fashion and not by broadcasting the corresponding subgraph.

**The Graph** $H$ Let $\ell$, $c$, and the skeleton graph $G_S = (S, E_S, \omega_S)$ be defined as in Section 18.3 and Equations (18.2)–(18.4), w.h.p. yielding $\mathrm{dist}(s,t,G_S) = \mathrm{dist}(s,t,G)$ for all $s,t \in S$. As in Section 18.3, however, we cannot afford to compute $G_S$. Suppose instead that we know $\omega'_S \colon S \times S \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ with

$$\forall s,t \in S\colon \quad \mathrm{dist}(s,t,G) \leq \omega'_S(s,t) \leq (1 + 1/\log^2 n)\, \omega_S(s,t); \tag{18.12}$$

we can determine $\omega'_S$ using $(1{+}1/\log^2 n)$-approximate $(S, \ell, |S|)$-detection [103].[4] Further, we can compute a $(d, \mathrm{O}(1/\log^2 n))$-hop set with $d \in n^{\mathrm{o}(1)}$ of $(S, E_S, \omega'_S)$ using the algorithm of Henzinger et al. [73]. Together, we may construct a graph

$$G'_S := (S, E'_S, \omega'_S), \tag{18.13}$$

where $E'_S$ contains the approximately weighted skeleton edges $E_S$ as well as the additional hop-set edges. $\omega'_S$ is extended to the hop-set edges and if $s,t \in S$ are connected by both a hop-set edge and a skeleton edge, we keep the shorter of the two. Then it w.h.p. holds for all $s,t \in S$ that

$$\mathrm{dist}(s,t,G_S) \leq \mathrm{dist}(s,t,G'_S) \leq \mathrm{dist}^d(s,t,G'_S) \in (1 + \hat{\varepsilon})\, \mathrm{dist}(v,w,G_S), \tag{18.14}$$

for some $\hat{\varepsilon} \in \mathrm{O}(1/\log^2 n)$. Next, embed $G'_S$ into $H_S$ as in Section 15.1, yielding node and edge levels $\lambda(e) \in \{0, \dots, \Lambda\}$:

$$H_S := \left(S, \binom{S}{2}, \omega_{H_S}\right) \text{ with} \tag{18.15}$$

$$\omega_{H_S}(\{s,t\}) := (1 + \hat{\varepsilon})^{\Lambda - \lambda(s,t)}\, \mathrm{dist}^d(s,t,G'_S). \tag{18.16}$$

By Theorem 15.5, we w.h.p. have that $\mathrm{SPD}(H_S) \in \mathrm{O}(\log^2 n)$ and for all $s,t \in S$ that

$$\mathrm{dist}(s,t,G_S) \leq \mathrm{dist}(s,t,H_S) \in \alpha\, \mathrm{dist}(s,t,G_S), \tag{18.17}$$

[4] $\omega'_S$ is not a $(1 + 1/\log^2 n)$-approximation of $\omega_S = \mathrm{dist}^{\ell}(\cdot,\cdot,G_S)$, because we might obtain distance estimations that are smaller than $\ell$-hop distances.

where an upper bound of $\alpha \in 1 + \mathrm{O}(1/\log n) \subseteq 1 + \mathrm{o}(1)$ can be derived analogously to Equation (15.18). Analogously to Equations (18.6)–(18.8), define

$$H := (V, E_H, \omega_H), \text{ where} \tag{18.18}$$

$$E_H := E \cup \binom{S}{2}, \text{ and} \tag{18.19}$$

$$\omega_H(e) := \begin{cases} \omega_{H_S}(e) & \text{if } e \in \binom{S}{2} \text{ and} \\ \alpha\, \omega_G(e) & \text{otherwise.} \end{cases} \tag{18.20}$$

By construction we thus have, for all $v, w \in V$ and w.h.p.,

$$\operatorname{dist}(v, w, G) \leq \operatorname{dist}(v, w, H) \leq \alpha \operatorname{dist}(v, w, G) \in (1 + \mathrm{o}(1)) \operatorname{dist}(v, w, G), \tag{18.21}$$

i.e., that $H$ is a metric embedding of $G$ with stretch $1 + \mathrm{o}(1)$.

**FRT Trees of $H$** Analogously to Section 18.3, we arrange that the node IDs of $S$ are ordered before those of $V \setminus S$. Hence, min-hop shortest paths in $H$ contain a single maximal subpath of edges in $E_{H_S}$: Distances in $H_S$ w.h.p. are $\alpha$-approximate distances of $G$, the original edges of $G$ are stretched by a factor of $\alpha$ in $H$, and hence subpaths of skeleton nodes w.h.p. are min-hop shortest paths in $H$. To determine the LE lists of $H$, we hence compute

$$r^V A_H^{\mathrm{SPD}(H)} x^{(0)} = \left(r^V A_{G,\alpha}\right)^{\ell} \underbrace{\left(r^V A_{H_S}\right)^{\mathrm{SPD}(H_S)} x^{(0)}}_{=:\bar{x}^{(0)}}, \tag{18.22}$$

where $A_{G,\alpha}$ is given by multiplying each entry of $A_G$ by the abovementioned factor of $\alpha$ and $A_{H_S}$ is extended to an adjacency matrix on the node set $V$ as in Section 18.3.

Let $\alpha' \in \mathrm{O}(\log n)$ denote the expected stretch of the FRT embedding. By construction, an FRT embedding of $H$ has an expected stretch of $\alpha\alpha' \in (1+\mathrm{o}(1))\alpha' \subseteq \mathrm{O}(\log n)$ w.r.t. $G$. In particular, the multiplicative overhead regarding the expected stretch is $(1 + \mathrm{o}(1))$, i.e., smaller than the multiplicative overhead of $\mathrm{O}(\varepsilon^{-1})$ induced by the spanner-based construction of Ghaffari and Lenzen [65].

**The Algorithm** We determine the LE lists of $H$ as follows, adapting the approach from [65] outlined in Section 18.3. Both algorithms have in common that they use $\bar{x}^{(0)}$ to jump-start the remaining $\ell \in \mathrm{O}(n^{1/2})$ iterations of Khan et al. [81]. The difference is that Ghaffari and Lenzen broadcast the skeleton spanner — resulting in the $n^{\varepsilon}$ overhead in the round complexity — and that we compute $\bar{x}^{(0)}$ in a distributed fashion.

(1) A node $v_0 \in V$ starts the computation by broadcasting a random choice of $\beta$. The broadcast is used to construct a BFS tree. Upon receipt of the broadcast, nodes generate random IDs of $\mathrm{O}(\log n)$ bits which are unique w.h.p. Analogously to Section 18.3, $v_0$ figures out the ID threshold of the bottom $|S|$ node IDs w.r.t. the induced random order using binary search. After broadcasting that threshold, every node knows whether it is in $S$. This can be done in $\tilde{\mathrm{O}}(\mathrm{D}(G))$ rounds.

(2) Each skeleton node $s \in S$ computes $\omega'_S(s,t)$ for all $t \in S$, using the $(1 + 1/\log^2 n)$-approximate $(S, \ell, |S|)$-detection algorithm by Lenzen and Patt-Shamir [103]. This takes $\tilde{\mathrm{O}}(\ell + n^{1/2} + \mathrm{D}(G)) \subseteq \tilde{\mathrm{O}}(n^{1/2} + \mathrm{D}(G))$ rounds.

(3) Run the algorithm of Henzinger et al. [73] to compute an $(n^{\mathrm{o}(1)}, \mathrm{O}(1/\log^2 n))$-hop set of $G'_S$. Here, all skeleton nodes learn all hop-set edges with the according weights; we only require, however, each $s \in S$ to learn its incident hop-set edges. This is possible in $\tilde{\mathrm{O}}((n^{1/2} + \mathrm{D}(G))n^{\mathrm{o}(1)})$ rounds by Theorem 4.7 in the full version of Henzinger et al. [73].

(4) Implicitly construct $H_S$. To this end, nodes in $S$ locally determine their level and broadcast it over the BFS tree, which takes $\mathrm{O}(|S| + \mathrm{D}(G)) \subset \tilde{\mathrm{O}}(n^{1/2} + \mathrm{D}(G))$ rounds.[5] Thus, $s \in S$ learns the level of $\{s,t\} \in E_{H_S}$ for each $t \in S$.

(5) To determine $\bar{x}^{(0)}$, we follow the same strategy as in Theorem 15.7, i.e., we simulate matrix–vector multiplication with $A_{H_S}$ via matrix–vector multiplications with $A_{G'_S}$ (where both matrices are extended to $V \times V$ matrices analogously to Section 18.3). Hence, it suffices to show that we can efficiently perform a matrix–vector multiplication $A_{G'_S}x$ for any $x$ that may occur during the computation — we may ignore applications of $r^V$ as it is a local operation and thus free — assuming each node $v \in V$ knows $x_v$ and its row of $A_{G'_S}$.

Since multiplications with $A_{G'_S}$ only affects lists at skeleton nodes, this can be done by local computations once all nodes know $x_s$ for each $s \in S$. We broadcast all $x_s$ of all skeleton nodes $s \in S$, rendering the remaining computations local.

As above, $|x_s| \in \mathrm{O}(\log n)$ w.h.p., so $\sum_{s \in S} |x_s| \in \mathrm{O}(|S| \log n) \subseteq \tilde{\mathrm{O}}(n^{1/2})$ w.h.p. We broadcast these lists over the BFS tree, taking $\tilde{\mathrm{O}}(n^{1/2} + \mathrm{D}(G))$ rounds per matrix–vector multiplication. Due to $\mathrm{SPD}(H_S) \in \mathrm{O}(\log^2 n)$ by Theorem 15.5 and $d \in n^{\mathrm{o}(1)}$, this results in a round complexity of $\tilde{\mathrm{O}}((n^{1/2} + \mathrm{D}(G))n^{\mathrm{o}(1)})$.

(6) Applying $r^V A^{\ell}_{G,\alpha}$ is analogous to step (3) in Section 18.3 and hence takes us an additional $\mathrm{O}(\ell \log n) \subseteq \tilde{\mathrm{O}}(n^{1/2})$ rounds.

Altogether, we arrive at a round complexity of $\tilde{\mathrm{O}}((n^{1/2} + \mathrm{D}(G))n^{\mathrm{o}(1)})$.

Recall that the algorithm of Khan et al. takes $\tilde{\mathrm{O}}(\mathrm{SPD}(G))$ rounds [81]. Hence it may be faster if $\mathrm{SPD}(G)$ is small. Using a standard technique, we can have the best of both worlds by running both algorithms simultaneously: Interleave the execution of both algorithms by, e.g., advancing one in even and the other in odd rounds, and return the result of whichever algorithm terminates first. We conclude the following result.

**Theorem 18.1.** *There is a randomized distributed algorithm in the* CONGEST *model that, in a graph $G$ with a polynomially bounded ratio between the minimum and maximum edge weight, w.h.p. computes a metric tree embedding of expected stretch* $\mathrm{O}(\log n)$ *using*

$$\min\left\{\tilde{\mathrm{O}}\left(\left(n^{1/2} + \mathrm{D}(G)\right) n^{\mathrm{o}(1)}\right), \tilde{\mathrm{O}}\left(\mathrm{SPD}(G)\right)\right\} \tag{18.23}$$

[5]This step can be merged with step (1) by having all nodes $v \in V$ sample a level $\lambda(v)$ independently from their random ID, $\mathrm{ID}(v)$. As $\lambda(v) \in \mathrm{O}(\log n)$ w.h.p. by Lemma 15.1, we can broadcast $(\mathrm{ID}(v), \lambda(v))$ instead of just $\mathrm{ID}(v)$. Later on, we only use the levels of skeleton nodes and ignore the other IDs. This, however, does not affect the asymptotic round complexity.

*rounds of computation. The expected stretch is larger than the expected stretch of an FRT embedding of $G$ by a factor of $1 + o(1)$.*

# CHAPTER 19

# Applications

In this chapter, we apply our parallel FRT embedding algorithm from Chapter 17 to $k$-median and the buy-at-bulk network design problem in Sections 19.1 and 19.2, respectively.

We improve upon the state of the art by generalizing the algorithm of Blelloch et al. [21] for $k$-median from constant-time query access metrics to arbitrary graphs, providing the same approximation guarantees. Regarding buy-at-bulk network design, we improve another algorithm by Blelloch et al. [21] by reducing the work from $\tilde{\mathrm{O}}(n^3)$ to $\tilde{\mathrm{O}}(\min\{m + n^{1+\varepsilon} + kn, n^2\}) \subseteq \tilde{\mathrm{O}}(n^2)$.

## 19.1 $k$-Median

Given a graph $G = (V, E, \omega)$, the $k$-median problem is about requests from clients located at a subset $V' \subseteq V$ of its nodes that are served by at most $k$ facilities $F \subseteq V$. For example, $G$ could represent a residential area, $V'$ customer locations, $\operatorname{dist}(\cdot, \cdot, G)$ walking distances, and $F$ the locations of convenience stores. Requests occur at client nodes $v \in V'$ and serving them incurs costs $\operatorname{dist}(v, F, G)$, the distance of $v$ to the closest facility $f \in F$:

$$\operatorname{dist}(v, F, G) := \min\{\operatorname{dist}(v, f, G) \mid f \in F\}. \tag{19.1}$$

The $k$-median problem takes the perspective of placing at most $k$ facilities in $G$, such that the sum of all possible requests is minimized.

**Definition 19.1** ($k$-Median)**.** *In the* $k$-median problem *we are given a weighted graph* $G = (V, E, \omega)$*,* clients $V' \subseteq V$*, and an integer* $k \in \mathbb{N}$*. The task is to determine* $F \subseteq V$ *with* $|F| \leq k$ *that minimizes*

$$\Phi\left(F, V', G\right) := \sum_{v \in V'} \operatorname{dist}(v, F, G). \tag{19.2}$$

Blelloch et al. study the $k$-median problem and give the following result [21]: Given a metric with constant-time query access, it is possible to determine an expected $\mathrm{O}(\log k)$-approximation of $k$-median using $\mathrm{O}(\log^2 n)$ depth and $\tilde{\mathrm{O}}(nk + k^3)$ work for $k \geq \log n$; the special case of $k < \log n$ admits an $\tilde{\mathrm{O}}(n)$-work solution of the same depth [22].

We improve upon this result using our techniques, the constraint being polylogarithmic depth. Our contribution is to accept an arbitrary weighted graph $G$ as input. This only provides implicit access to the distance metric $\operatorname{dist}(\cdot, \cdot, G)$, instead constant-time query access. Hence, our solution is more general as any finite metric defines a complete graph of SPD 1. The use of hop sets, however, restricts us to polynomially bounded edge-weight ratios. Below, we establish that the same approximation guarantees provided by Blelloch et al. — w.h.p. an expected $\mathrm{O}(\log k)$-approximation for $k \geq \log n$ and an $\mathrm{O}(1)$-approximation for $k < \log n$ — of $k$-median can be achieved in a weighted graph using $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon} + k^3)$ work.

Observe that a weaker version of our result that uses $\tilde{\mathrm{O}}(n^{2+\varepsilon} + k^3)$ work directly follows from (1) applying Corollary 16.6 to obtain a metric that $\mathrm{O}(1)$-approximates $\operatorname{dist}(\cdot, \cdot, G)$ and (2) using the algorithm of Blelloch et al. [21] as black box on that metric. Below, we show how to reduce the work to $\tilde{\mathrm{O}}(m^{1+\varepsilon} + k^3)$. This, however, requires us to adapt some steps of the existing algorithm.

Let $G = (V, E, \omega)$ be a weighted graph, $V' \subseteq V$ a set of clients, and $k \in \mathbb{N}$ an integer. Recall that Blelloch et al. require constant-time query access to $\operatorname{dist}(\cdot, \cdot, G)$ and let $\mathrm{OPT} := \min\{\Phi(F, V', G) \mid F \subseteq V, |F| \leq k\}$ denote the cost of an optimal $k$-median solution w.r.t. the above instance. The algorithm of Blelloch et al. [21] comprises the following steps:

(1) Use a parallel version of a sampling technique due to Mettu and Plaxton [112] (Section 4.4 of Blelloch et al. [21]). It samples candidates $Q$ that guarantee the following properties: (a) $\Phi(Q, V', G) \leq \beta \,\mathrm{OPT}$ w.h.p. for some $\beta \in \mathrm{O}(1)$ and (b) $q := |Q| \in \mathrm{O}(K \log \frac{n}{K})$, where $K = \max\{k, \log n\}$. Note, however, that $Q$ may be infeasible for the original instance due to $q > k$.

    In the following, associate each client $v \in V'$ with a center $q \in Q$ that minimizes $\operatorname{dist}(v, q, G)$. Furthermore, consider the weighted $k$-median problem on the submetric spanned by $Q$, where each $q \in Q$ is weighted with the number of clients in $V'$ associated with it; if $q \in Q$ receives weight 0, we remove it from the set of clients. This guarantees that an $\alpha$-approximation of the weighted instance w.h.p. $\alpha\beta$-approximates OPT (Appendix A.1 of Blelloch et al. [21]).

(2) If $k < \log n$, solve the modified instance exactly, using the algorithm of Blelloch and Tangwongsan [22], and return the result. On an input of size $n$, the algorithm of Blelloch and Tangwongsan uses $\mathrm{O}(\log^2 n)$ depth and $\mathrm{O}(k^2 n^2 \log n)$ work. Due to $k < \log n$, we have $K = \log n$ and $q \in \mathrm{O}(\log^2 n)$, hence, the work is $\mathrm{O}(k^2 q^2 \log n) \subseteq \mathrm{O}(\log^7 n)$. This special case yields an $\mathrm{O}(1)$-approximation.

(3) If $k \geq \log n$, we have $K = k$ and hence $q \in \mathrm{O}(k \log \frac{n}{k})$. Sample an FRT tree regarding the submetric spanned by $Q$. Normalize the tree to a binary tree — this is required by the next step — which is possible without incurring too much overhead w.r.t. the depth of the tree (Section 4.2 of Blelloch et al. [21]).

(4) Run an $\mathrm{O}(kq^2)$-work dynamic-programming algorithm to solve the weighted binary tree instance optimally without using any Steiner nodes (Sections 4.1 and 4.3 of Blelloch et al. [21]). In expectation, this w.h.p. yields an $\mathrm{O}(\log q) \subseteq \mathrm{O}(\log k^2) \subseteq \mathrm{O}(\log k)$-approximate solution of the original problem instance by the observation in step (1).

Blelloch et al. show how the above steps can be performed using $\mathrm{O}(nk + k^3) \subseteq \mathrm{O}(kn^2)$ work and $\mathrm{O}(\log^2 n)$ depth, w.h.p. guaranteeing a constant-factor approximation in the case of $k < \log n$ and an expected $\mathrm{O}(\log k)$-approximation if $k \geq \log n$ [21].

We leave the overall structure in place and modify steps (1)–(3). The main challenge is to account for the lack of constant-time distance queries due to accepting a general graph as input. We propose the following algorithm:

(1) Augment $G$ with a $(d, \hat{\varepsilon})$-hop set. This is possible such that we obtain $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ edges and $d, \hat{\varepsilon}^{-1} \in \operatorname{polylog} n$, using polylog $n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work [34]. Observe that this does not change the approximation quality.

(2) The sampling procedure — step (1) in the above algorithm — uses $\mathrm{O}(\log \frac{n}{K})$ iterations. It maintains a candidate set $U$ that initially contains all vertices. In each iteration, $\mathrm{O}(K)$ candidates $S$ are sampled and added to $Q$. Then, a constant fraction of vertices in $U$, those closest to $S$, is removed from $U$ (Algorithm 4.1 of Blelloch et al. [21]). After the last iteration, all vertices remaining in $U$ are added to $Q$.

The key to adapting this procedure lies in efficiently determining $\operatorname{dist}(u, S, G)$ for all $u \in U$ in order to determine which subset of $U$ to discard (this is trivial with constant-time query access to the metric).

We achieve this by sampling in $H$ from Chapter 15 which only costs a factor of $(1 + \mathrm{o}(1))$ in approximation, regardless of $k$. By Theorem 15.5, we only require $\mathrm{O}(\log^2 n)$ iterations of the MBF-like algorithm from Example 14.8 for $d = \infty$ — this is $(S, \mathrm{SPD}(H), \infty, 1)$-source detection — to determine each node's distance to the closest vertex in $S$ w.h.p. After that, binary search reveals the distance threshold and with it the nodes to be discarded.

An iteration of $(S, \mathrm{SPD}(H), \infty, 1)$-source detection is simple; each node only has to pick the smallest distance received from its neighbors. Hence, an iteration only takes $D \in \mathrm{O}(\log n)$ depth and work $W \in \mathrm{O}(m^{1+\varepsilon} \log n)$, aggregation over $\Lambda \in \mathrm{O}(\log n)$ values takes $D_{\oplus} \in \mathrm{O}(\log n)$ depth and $W_{\oplus} \in \mathrm{O}(\log^2 n)$ work. Theorem 15.7 hence allows us to run the algorithm in $H$ using polylog $n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work. As there are $\mathrm{O}(\log n)$ iterations, we can afford to do this in every iteration.

The other parts of this step can be done analogously to Blelloch et al. [21].

(3) If $k < \log n$ we can afford to determine a metric that $(1 + \mathrm{o}(1))$-approximates $(\operatorname{dist}(v, w, G))_{v,w \in Q}$ using polylog $n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon})$ work; this follows from Corollary 16.5 with $|S| = |Q| \in \mathrm{O}(\log^2 n)$ (compare step (2) of the above algorithm). As the approximate metric offers constant-time query access, we may apply the algorithm of Blelloch and Tangwongsan [22], within the stated limits of depth and work and immediately return the result. Further, as we only increase the stretch by a factor in $(1 + \mathrm{o}(1))$, we obtain an $\mathrm{O}(1)$-approximation analogously to step (2) of the above algorithm.

(4) Otherwise, we have $k \geq \log n$ and sample an FRT tree on the submetric implicitly spanned by $Q$ in $H$. To compute the embedding only on $Q$ set $x^{(0)}_{vv} = 0$ if $v \in Q$ and $x^{(0)}_{vw} = \infty$ everywhere else. Consider only the LE lists of nodes in $Q$ when constructing the tree. As we are limited to polynomially bounded edge-weight ratios, our FRT trees have logarithmic depth. We normalize to a binary tree using the same technique as Blelloch et al. [21].

(5) The $\mathrm{O}(kq^2) \subseteq \mathrm{O}(k(k \log \frac{n}{k})^2) \subseteq \mathrm{O}(k^3 \log^2 n) \subseteq \tilde{\mathrm{O}}(k^3)$-work polylogarithmic-depth dynamic-programming algorithm of Blelloch et al. can be applied without modification.

W.h.p., we arrive at an expected $\mathrm{O}(\log k)$-approximation of $k$-median for $k \geq \log n$ and a constant-factor approximation if $k < \log n$.

**Theorem 19.2.** *For any fixed constant $0 < \varepsilon \leq \frac{1}{2}$, there is an algorithm that w.h.p. computes an expected $\mathrm{O}(\log k)$-approximation to $k$-median in a weighted graph using $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m^{1+\varepsilon} + k^3)$ work. The special case of $k < \log n$ yields an $\mathrm{O}(1)$-approximation.*

## 19.2 Buy-at-Bulk Network Design

In the buy-at-bulk network design problem, we consider a weighted graph $G = (V, E, \omega)$ and are given several demands, i.e., source–target node pairs together with a requested capacity. Furthermore, there are different types of cables available, each type has a capacity and a price per unit. The task is to place enough cables on the edges of the graph, where the length of an edge denotes how much of the cable has to be bought, such that all demands can be served simultaneously. It is possible to buy the same type of cable several times for the same edge. Possible applications of this problem include, as indicated by its name, the design of computer or power-supply networks.

**Definition 19.3** (Buy-at-Bulk Network Design)**.** *In the* buy-at-bulk network design problem, *one is given*

*(1)* *a weighted graph $G = (V, E, \omega)$,*

*(2)* demands $(s_i, t_i, d_i) \in V \times V \times \mathbb{R}_{>0}$ *for $1 \leq i \leq k$, and*

*(3)* cable *types $(u_i, c_i) \in \mathbb{R}_{>0} \times \mathbb{R}_{>0}$ for $1 \leq i \leq \ell$, where $u_i$ is the capacity and $c_i$ the cost per unit of length.*

*The goal is to choose exactly one $s_i$-$t_i$-path — we may not split the flow — for each demand and to find an assignment of cable types and multiplicities to edges minimizing the total cost, such that the resulting edge capacities allow to route $d_i$ units of distinct flow from $s_i$ to $t_i$ for all $1 \leq i \leq k$ simultaneously. The purchase of a cable of type $i$ for the edge $e$ incurs a cost of $c_i\,\omega(e)$. Multiple cables of the same type can be bought for an edge.*

Blelloch et al. give an expected $\mathrm{O}(\log n)$-approximation w.h.p. using $\operatorname{polylog} n$ depth and $\mathrm{O}(n^3 \log n)$ work for the buy-at-bulk network design problem; in the case constant-time query access to $\operatorname{dist}(\cdot, \cdot, G)$ is available, the work is reduced to $\mathrm{O}(n^2 \log n)$ because solving APSP dominates the work [21].

Observe that using Corollary 16.6 to $\mathrm{O}(1)$-approximately solve APSP while preserving the triangle inequality, we immediately arrive, w.h.p., at a $\tilde{\mathrm{O}}(n^{2+\varepsilon})$-work algorithm of $\operatorname{polylog} n$ depth with the same asymptotic approximation guarantee as Blelloch et al.

Our tools, however, admit a more work-efficient parallelization that w.h.p. uses $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(\min\{m+n^{1+\varepsilon}+kn, n^2\}) \subseteq \tilde{\mathrm{O}}(n^2)$ work, without requiring constant-time query access to $\operatorname{dist}(\cdot, \cdot, G)$. In expectation, we achieve a $\mathrm{O}(\log n)$-approximation, i.e., the same asymptotic guarantee as Blelloch et al. [21]. The use of hop sets, however, restricts us to polynomially bounded edge-weight ratios.

We propose the following modification of the approach of Blelloch et al. [21]:

(1) Metrically embed $G$ into tree $T = (V_T, E_T, \omega_T)$ with an expected stretch of $\mathrm{O}(\log n)$ with $V_T = V$, see Observation 17.17. Dispensing with Steiner nodes is required by the underlying approximation algorithm of Awerbuch and Azar [7].

(2) $\mathrm{O}(1)$-approximate in $T$ by picking, for each $e \in E_T$, the cable of type $i$ that minimizes $c_i \lceil d_e/u_i \rceil$, where $d_e$ is the accumulated flow on $e$. Awerbuch and Azar argue why it is a constant-factor approximation [7].

(3) Map the tree solution back to $G$ as detailed in Observations 17.14–17.16.

First observe that the above procedure yields an expected $\mathrm{O}(\log n)$-approximation: Let $\mathrm{OPT}_G$ and $\mathrm{OPT}_T$ denote the costs of the respective optimal solutions in $G$ and in $T$. Translating the optimal solution of $G$ to $T$ yields, in expectation, costs of $\beta \,\mathrm{OPT}_G$ for some $\beta \in \mathrm{O}(\log n)$ due to the expected stretch of the tree embedding and the fact that the objective function is linear in the edge weights. As we $\alpha$-approximate in $T$, where $\alpha \in \mathrm{O}(1)$, and $\mathrm{OPT}_T \leq \beta \,\mathrm{OPT}_G$, our solution in $T$ costs at most

$$\alpha \,\mathrm{OPT}_T \leq \alpha\beta \,\mathrm{OPT}_G \in \mathrm{O}(\log n)\,\mathrm{OPT}_G \tag{19.3}$$

in $T$. We map our tree solution back to $G$ without incurring additional costs.

Using Corollary 17.13, the first step has $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(m + n^{1+\varepsilon})$ work. For the second step, Blelloch et al. discuss an algorithm of $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(n + k)$ work [21]. Concerning the third step, observe that we can map a leaf-leaf path in $T$ to a path of $\tilde{\mathrm{O}}(n)$ hops in $G$ — possibly with loops — using Observations 17.14–17.16 and the fact that $T$ has depth $\mathrm{O}(\log n)$ due to the polynomially bounded edge-weight ratios. Since there are $k$ demand pairs, we evaluate $\min\{k, n\}$ such paths ($T$ only has $n$ leaves), resulting in $\tilde{\mathrm{O}}(\min\{kn, n^2\})$ work. As discussed in Observations 17.14–17.16, we assume that appropriate data structures for doing this using $\operatorname{polylog} n$ depth are in place. The above algorithm hence has

$$\tilde{\mathrm{O}}\left(m + n^{1+\varepsilon} + n + k + \min\left\{kn, n^2\right\}\right) \subseteq \tilde{\mathrm{O}}\left(\min\left\{m + n^{1+\varepsilon} + kn, n^2\right\}\right) \tag{19.4}$$

work and $\operatorname{polylog} n$ depth. We arrive at the following theorem.

**Theorem 19.4.** *For any constant $0 < \varepsilon \leq \frac{5}{4}$, w.h.p., an expected $\mathrm{O}(\log n)$-approximation to the buy-at-bulk network design problem can be computed using $\operatorname{polylog} n$ depth and $\tilde{\mathrm{O}}(\min\{m + n^{1+\varepsilon} + kn, n^2\}) \subseteq \tilde{\mathrm{O}}(n^2)$ work.*

# CHAPTER 20
# Conclusion

We propose an algebraic classification of MBF-like algorithms in Chapter 13. This is a new perspective on a large group of algorithms and formalizes the characterization of "algorithms that behave like the MBF algorithm." Besides being of theoretical interest, our characterization captures many algorithms. We demonstrate this in Chapter 14 and reduce the task of devising and analyzing such algorithms to the following recipe:

(1) Pick a suitable semiring $\mathcal{S}$ and semimodule $\mathcal{M}$ over $\mathcal{S}$. While custom semirings and semimodules can be used, our hope is that the generic variants proposed in Chapter 14 prove useful for many algorithms.

(2) Choose a filter $r\colon \mathcal{M} \to \mathcal{M}$, initial values $x^{(0)} \in \mathcal{M}^V$, and a number of iterations $h \in \mathbb{N}_0$, such that $r^V A^h x^{(0)}$ is the desired output. The filter $r$ is where the problem-specific customization happens. Hence, we hope that a small number of generic semimodules can be repurposed for many algorithms.

(3) Verify that $r$ induces a congruence relation on $\mathcal{M}$.

(4) Leverage repeated use of $r^V$ to ensure that iterations can be implemented efficiently. This is where irrelevant information — in the sense that some part of the accumulated information is irrelevant not only in the current but also in future iterations — is dropped, as soon as it is classified as such.

We focus on parallel, polylogarithmic-depth, low-work algorithms.

A key application of the proposed class of MBF-like algorithms is that it allows formalizing a statement like "one iteration of an algorithm in the graph $H$ is equivalent to $d$ appropriately aggregated iterations in the graph $G$." Without a generic framework, this has to be done on a per-algorithm basis; in our case it would have entangled the LE list computation with the graph $H$, making it inconvenient at best to show that the same approach also works for parallel metric approximations.

The framework allows us to devise a simulation strategy for MBF-like algorithms over the $\mathcal{S}_{\min,+}$ semiring in Chapter 15, where we are able to devise a graph $H$ of polylogarithmic SPD — which is useful for polylogarithmic-depth parallel algorithms — that (1) we cannot explicitly compute because it is complete and thus induces too much work, but that (2) allows simulating MBF-like algorithms in $H$ using only iterations in the original graph $G$ and polylogarithmic overhead in depth and work. Together with an appropriate hop-set algorithm, this allows us to devise the oracle with the property that if we can run *a single iteration* (and other simple operations) of an arbitrary MBF-like algorithm $\mathcal{A}$ over $\mathcal{S}_{\min,+}$ in $G$ using $D$ depth and $W$ work, then we can simulate *all iterations* of $\mathcal{A}$ in $H$ using $\tilde{\mathrm{O}}(D)$ depth and $\tilde{\mathrm{O}}(W)$ work. As distances in $H$ $(1+\mathrm{o}(1))$-approximate distances in $G$, the oracle yields a good approximation of distances that at the same time are *consistent with the triangle inequality* at only a polylogarithmic overhead.

It is the consistency with the triangle inequality that allows us to determine approximate metrics and submetrics in Chapter 16, because MSSP and APSP are MBF-like. Further, collecting LE lists — the key part of sampling an FRT tree — is MBF-like, allowing us to determine an FRT-style metric tree embedding of expected logarithmic stretch in Chapter 17.

While we consider the polylogarithmic factors too large for our algorithm to be of practical interest, this result motivates the search for solutions that achieve low depth despite having work comparable to the currently best known sequential bound of $\mathrm{O}(m \log n)$ w.h.p. [20]. Concretely, better hop-set constructions could readily be plugged into our machinery to yield improved bounds and one may seek to reduce the polylogarithmic overhead incurred by the remaining construction.

Further, we use our techniques to improve upon the stretch and the round complexity required to sample an FRT-style embedding in the CONGEST model in Chapter 18 and to improve upon the state of the art of parallel approximation algorithms regarding the $k$-median and the buy-at-bulk network design problem in Chapter 19.

We hope that our framework of MBF-like algorithms, the oracle, and our parallel FRT construction are useful in the design of parallel and distributed algorithms.

A possible generalization of the framework of MBF-like algorithms is to not propagate the entire filtered knowledge of a node, but just a subset thereof. This is sometimes required in the CONGEST model to ensure that the maximum message size is not exceeded.

# PART III

# Terrain Guarding

*"There was a terrible pseudo-familiarity about them — which somehow made me look furtively and apprehensively over the abominable, sterile terrain toward the north and northeast."*

H. P. Lovecraft, The Shadow out of Time

This part is based on the article that appeared in the *7th volume of the Journal of Computational Geometry (JoCG),* pages 256–284, 2016 [56] which extends and subsumes Chapter 3 of *James King's PhD thesis,* pages 29–72, 2010 [86], the extended abstracts that appeared in the *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG 2014),* pages 367–373, 2014 [57] and in the *31st European Workshop on Computational Geometry (EuroCG 2015),* pages 212–215, 2015 [58], and the *Terrain Guarding Problem Instance Library* [59].

The following chapters are the result of close collaboration with Michael Hemmer and Christiane Schmidt. James King published the first discretization of the Continuous Terrain Guarding Problem [86]. The discretization developed with Michael Hemmer and Christiane Schmidt in [57, 58] was discovered independently from, and unaware of, James King's [86]. Our subsequent joint publication [56] contains the discretization of Friedrichs et al. [57, 58]; it beats the asymptotic complexity of that of James King [86] by a factor of $n$.

# CHAPTER 21

# Introduction

In the 1.5D Terrain Guarding Problem (TGP), we are given an $x$-monotone chain of line segments in $\mathbb{R}^2$, the *terrain* $T$. We want to place a minimum number of point-shaped guards on $T$ that together cover $T$, where $g \in T$ covers $w \in T$ if and only if the line segment $\overline{gw}$ is nowhere below $T$, compare Figure 21.1(a). Our main interest is the Continuous Terrain Guarding Problem (CTGP), where guards can be placed anywhere on $T$. The CTGP is more challenging than TGP versions with an a priori discretization, e.g., the Terrain Guarding Problem with Vertex Guards (VTGP): Good guard candidates are not necessarily located on the vertices, compare Figure 21.1(b), and have to be identified first.

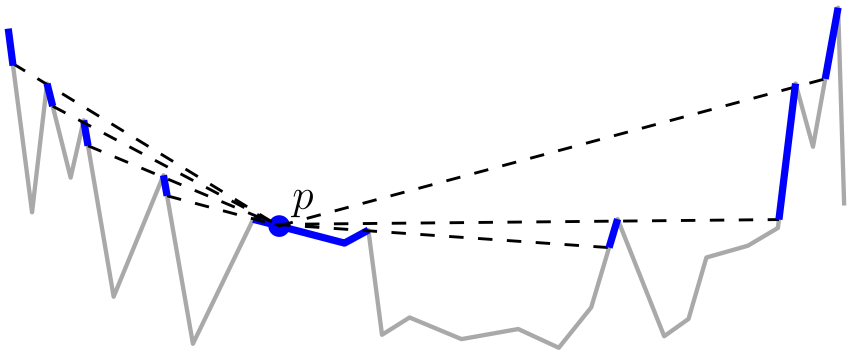
The TGP is a close relative of the Art Gallery Problem (AGP), where a minimum number of point-shaped guards is to be placed to cover[1] a polygon. Traditionally, the TGP is motivated by the optimal placement of antennas for line-of-sight communication networks, and the placement of street lights or security cameras along roads [18]. We would, however, like to revive a motivation rooted in our research regarding algorithms solving the AGP [40, 49, 51] already mentioned by Ben-Moshe et al. [18]: An application of the AGP is the placement of sensors or communication devices w.r.t. obstacles, for example placing laser scanners in production facilities to acquire a precise mapping of the facility [49, 95].

While the AGP properly models most indoor environments, it cannot capture many outdoor scenarios, like placing cell phone towers in an urban environment, because it does not take height information into account. Fixing this requires two dimensions and height, a 2.5D AGP. In order to study the aspect of elevation, one dimension and height, the 1.5D TGP, is a natural starting point to develop techniques for the 2.5D AGP. Despite being a simplification of the AGP, the CTGP is NP-hard [87] and thus not an oversimplification.

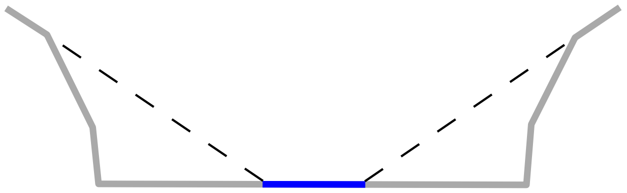
We show that the "height dimension" of the TGP is more benevolent than the "second dimension" of the AGP: 1.5D terrains with rational vertex coordinates have a polynomial-size discretization of only rational coordinates. This is surprising, because Abrahamsen et al. recently showed that this is not the case for the AGP: Guards with irrational coordinates are sometimes required to cover polygons with rational vertex coordinates [1]. This is a serious issue for software solving the AGP, where the discretization of subproblems poses a key challenge [40], and by extension the 2.5D AGP. Nevertheless, we hope that our result indicates that the 2.5D AGP is not harder than the AGP.

Another motivation for discretizing the CTGP is the following. The existence of a polynomial-size discretization immediately yields a Polynomial-Time Approximation Scheme (PTAS) for the CTGP and establishes the NP-completeness of its decision variant. This follows because the discrete TGP is known to permit a PTAS [66] and the decision

[1]Analogously to the TGP, a guard $g$ covers the point $w$ in a polygon $P$ if and only if $\overline{gw} \subseteq P$.



**(a)** The visibility region $\mathcal{V}(p)$ of $p \in T$ has $O(n)$ subterrains.



**(b)** This terrain needs two vertex- but only one non-vertex guard [18].

**Figure 21.1:** The Terrain Guarding Problem.

variant of the CTGP is known to be NP-hard [87]. Unknown to the authors of Friedrichs et al. [57, 58] until after publication, King developed a discretization of size $O(n^4)$ [86]. The discretization independently developed by the authors is presented in Chapter 22. It has size $O(n^3)$, i.e., is asymptotically smaller by a factor of $n$ and is the foundation of the joint publication [56]. Hence, the PTAS and the NP-completeness of the CTGP in Chapter 23, our initial motivation, are implicit to prior work.

## 21.1 Our Contribution

(1) Our core contribution is Chapter 22. We show that the CTGP, where guards can be freely placed on the terrain, has a discretization with $O(n^2)$ guard candidates[2] and $O(n^3)$ witnesses;[3] the discretization only requires rational coordinates for terrains restricted to rational vertex coordinates. Our discretization is asymptotically smaller than that of King [86] by a factor of $n$ and was discovered independently (see above). Furthermore, we propose filtering techniques that can drastically reduce the size of the discretization and thus pave the way for an efficient implementation.

(2) The results of Chapter 23 immediately follow.

   (a) While the decision version of the CTGP is known to be NP-hard [87], we conclude that it also is a member of NP and hence NP-complete.

[2]Possible guard locations.

[3]Coverage of the entire terrain follows from coverage of all witness points by guard candidates.

   (b) It follows from the PTAS for the discrete TGP of Gibson et al. [66] that there is a PTAS for the CTGP.

   As discussed above, both results are implicit to prior work [86].

(3) An efficient algorithm for continuous and discrete TGP versions is proposed in Chapter 24. It is based on an Integer Linear Program (IP) and finds optimal solutions for terrains with up to $10^6$ vertices on a standard desktop computer[4] within minutes. This is achieved following the Exact Geometric Computation (EGC) paradigm, i.e., using exact arithmetic for geometric calculations to ensure correctness. We test our algorithm and the filtering techniques from Chapter 22 on a large collection of instances [59].

We conclude this part in Chapter 25.

## 21.2 Related Work

The TGP is closely related to the AGP, where, given a polygon $P$, we seek a minimum cardinality guard set that covers $P$. Potential guards can, e.g., be located on the vertices only, on arbitrary points in $P$, or patrol along edges or diagonals of $P$. Many polygon classes have been considered for the AGP, including simple polygons, polygons with holes, and orthogonal polygons. Moreover, the guards' task can be altered, e.g., Laurentini [97] required visibility coverage for the edges of $P$, but not the interior.

The first result in the context of the AGP was obtained by Chvátal [30], who proved the *Art Gallery Theorem,* answering a question posed by Victor Klee in 1973, see O'Rourke [117]: $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and sometimes necessary to guard a polygon of $n$ vertices. A simple and elegant proof of the sufficiency was later given by Fisk [53]. Related results were obtained for various polygon classes; Kahn et al. [79] established a tight bound of $\lfloor \frac{n}{4} \rfloor$ for orthogonal polygons with $n$ vertices.

The above results focus on bounds regarding the number of guards. However, the AGP also is an optimization problem: Given a polygon, find a minimum number of guards covering it. The decision variant of this optimization problem was shown to be NP-hard for various problem versions [118, 124], even for vertex guards in polygons without holes [98]. Eidenbenz et al. established APX-hardness of many AGP variants [45]. Chwa et al. [31] considered witnessable polygons, in which coverage of some finite set of witness points implies coverage of the entire polygon. For classical surveys on the AGP see O'Rourke [117] and Shermer [125], and de Rezende et al. [40] for more recent computational developments.

For the 1.5D TGP, research first focused on approximation algorithms, because NP-hardness was generally assumed, but had not been established. The first constant-factor approximation was given by Ben-Moshe et al. [18] for the discrete vertex guard problem version $\mathrm{TGP}(V,V)$,[5] where only vertex guards are used to cover only the vertices. They were able to use it as a building block for an $\mathrm{O}(1)$-approximation of $\mathrm{TGP}(T,T)$, where guards on arbitrary locations on $T$ must guard all of $T$. The approximation factor

[4]Standard as of 2015: An Intel Core i7-3770 CPU with 3.4 GHz with 14 GB of main memory.

[5]$\mathrm{TGP}(G,W)$ means that $W \subseteq T$ must be covered using only guards located in $G \subseteq T$, see Definition 21.1. CTGP refers to $\mathrm{TGP}(T,T)$ and VTGP to $\mathrm{TGP}(V,T)$.

of this algorithm was not stated by the authors, but claimed to be 6 by King [85] (with minor modifications). Another constant-factor approximation based on $\varepsilon$-nets and Set Cover (SC) was given by Clarkson and Varadarajan [32]. King [85] presented a 4-approximation (which was later shown to actually be a 5-approximation [84]) for $\mathrm{TGP}(V, V)$ and $\mathrm{TGP}(T, T)$. The most recent $\mathrm{O}(1)$-approximation was presented by Elbassioni et al. [46]: Using LP-rounding techniques, they achieve a 4-approximation of $\mathrm{TGP}(T, T)$ and $\mathrm{TGP}(G, W)$ w.r.t. finite, disjoint $G, W \subset T$ (a 5-approximation if $G \cap W \neq \emptyset$). This approximation generalizes to the TGP with weighted[6] guards. In the 2009 conference version, Gibson et al. [66] devised a PTAS based on local search for $\mathrm{TGP}(G, W)$ and $\mathrm{TGP}(G, T)$, where $G, W \subset T$ are finite.

Only after all these approximation results, in the 2010 conference version, King and Krohn [87] established the $\mathsf{NP}$-hardness of both the discrete and the continuous TGP by a reduction from PLANAR 3SAT. The membership of the CTGP in $\mathsf{NP}$ was implicitly established by King [86], but, to the best of our knowledge, not published in conference or journal articles before our joint publication [56]. Khodakarami et al. [82] showed that the TGP is Fixed-Parameter Tractable (FPT) w.r.t. the *depth of terrain onion peeling,* the number of layers of upper convex hulls induced by a terrain.

Variants of the TGP include guards hovering above the terrain (Eidenbenz [44]), orthogonal terrains (Katz and Roisman [80]), and directed visibility (Durocher et al. [42]). Hurtado et al. [75] gave algorithms for computing visibility regions in 1.5D and 2.5D terrains. Haas and Hemmer [70] presented implementations for 1.5D visibility based on Hurtado et al. [75] and the triangular expansion technique for visibility computations in polygons by Bungiu et al. [27].

Martinović et al. [109] proposed an approximate solver for the discrete TGP. Requiring a priori knowledge about pairwise visibility of the vertices $V$, they 5.5- and 6-approximate $\mathrm{TGP}(V, V)$ on instances with up to 8000 vertices and dense (0.19–0.65) visibility matrices in 11–900 and 4–250 seconds, respectively. As visibility information is encoded in the input, they are not tied to the EGC paradigm; hence, they use floating-point arithmetic and a parallel GPU implementation. Note that we solve a different problem: We determine the discretization and visibility information that Martinović et al. require as input, follow the EGC paradigm, and guarantee optimal solutions in no more than 3.5 seconds for 10000 vertices. However, we do not focus on dense visibility matrices and use different hardware, rendering a comparison of computation times meaningless.

Regarding a discretization for the CTGP, Gibson et al. claimed that their local search works well, but could not bound the number of bits representing the guard positions [66]. King gave a discretization with $\mathrm{O}(n^3)$ guard candidates and $\mathrm{O}(n^4)$ witnesses in his PhD thesis in 2010 [86] and posed the question whether a smaller discretization exists. Independently, Friedrichs et al. discovered a discretization using $\mathrm{O}(n^2)$ guard candidates and $\mathrm{O}(n^3)$ witnesses [57, 58] in 2014; there was a joint publication in 2016 [56]. This work subsumes and extends the joint publication [56], as detailed above. Recently, Abrahamsen et al. showed that the AGP sometimes requires irrational guards [1]. This makes the existence of a rational discretization for the CTGP surprising, especially because the proof of Abrahamsen et al. only requires monotone polygons which — unlike arbitrary

[6]Guards with non-uniform cost.

polygons — are subject to many of the structural restrictions of 1.5D terrains.

## 21.3 Notation and Preliminaries

A *terrain* $T$, see Figure 21.1, is an $x$-monotone chain of line segments in $\mathbb{R}^2$ defined by its *vertices* $V(T) = \{v_1, \dots, v_n\}$ that has *edges* $E(T) = \{e_1, \dots, e_{n-1}\}$ with $e_i := \overline{v_i v_{i+1}}$. Unless specified otherwise, $n := |V(T)|$. Where $T$ is clear from context, we occasionally abbreviate $V(T)$ and $E(T)$ by $V$ and $E$. $v_i$ and $v_{i+1}$ are the vertices of the edge $e_i$, and $\text{int}(e_i) := e_i \setminus \{v_i, v_{i+1}\}$ is its *interior.* Observe that all edges $e_i$ are closed sets, $v_i, v_{i+1} \in e_i$, while $\text{int}(e_i)$ is not closed. Due to monotonicity, the points on $T$ are totally ordered w.r.t. their $x$-coordinates. For $p, q \in T$, we write $p \leq q$ $(p < q)$ if $p$ is (strictly) left of $q$, i.e., has a (strictly) smaller $x$-coordinate. We refer to a closed, connected subset of $T$ as a *subterrain.*

A point $p \in T$ *sees* or *covers* $q \in T$ if and only if $\overline{pq}$ is nowhere below $T$. With

$$\mathcal{V}(p) := \{q \in T \mid p \text{ sees } q\} \tag{21.1}$$

we denote the *visibility region* of $p$. Observe that $\mathcal{V}(p)$ is the union of $\mathrm{O}(n)$ subterrains, see Figure 21.1(a), hence, $\mathcal{V}(p)$ is closed but not necessarily connected. We say that $q \in \mathcal{V}(p)$ is *extremal in* $\mathcal{V}(p)$ if $q$ has a maximal or minimal $x$-coordinate within its subterrain in $\mathcal{V}(p)$. For $G \subseteq T$ we abbreviate

$$\mathcal{V}(G) := \bigcup_{g \in G} \mathcal{V}(g). \tag{21.2}$$

A set $G \subseteq T$ with $\mathcal{V}(G) = T$ is a *(guard) cover* of $T$. In this context, $g \in G$ is sometimes referred to as *guard.*

**Definition 21.1** (Terrain Guarding Problem)**.** *In the* Terrain Guarding Problem (TGP) *we are given a terrain $T$ and sets of guard candidates and witnesses $G, W \subseteq T$. We abbreviate this by* $\text{TGP}(G, W)$*, omitting $T$ as an explicit parameter because it is clear from context throughout the following chapters. $C \subseteq G$ is* feasible w.r.t. $\text{TGP}(G, W)$ *if and only if $W \subseteq \mathcal{V}(C)$. We define*

$$\text{OPT}(G, W) := \min\{|C| \mid C \subseteq G \textit{ is feasible w.r.t. } \text{TGP}(G, W)\} \tag{21.3}$$

*and call $C \subseteq G$* optimal w.r.t. $\text{TGP}(G, W)$ *if and only if $C$ is feasible and $|C| = \text{OPT}(G, W)$. $\text{TGP}(G, W)$ asks for an optimal guard cover $C \subseteq G$. The* Continuous Terrain Guarding Problem (CTGP) *is* $\text{TGP}(T, T)$ *and the* Terrain Guarding Problem with Vertex Guards (VTGP) *is* $\text{TGP}(V(T), T)$*.*

Observe that $\text{TGP}(T, T)$ is well-defined: Placing a guard on every other vertex is feasible and yields $\text{OPT}(T, T) \leq \frac{n}{2}$. In the following, we assume $W \subseteq \mathcal{V}(G)$, i.e., that $\text{TGP}(G, W)$ has a feasible solution. The CTGP is our primary focus. Observe that CTGP and VTGP are different problems [18], as demonstrated in Figure 21.1(b). We consider VTGP a representative of the numerous discrete versions of the TGP; our algorithm solves both CTGP and VTGP, and generalizes to arbitrary discretizations.

# CHAPTER 22

# Discretization

This chapter is the foundation of our contributions to the CTGP. We consider the following problem: Given a terrain $T$ with $n$ vertices, construct sets $G, W \subset T$ (guard candidate and witness points) of size polynomial in $n$, such that any feasible (optimal) solution for $\mathrm{TGP}(G, W)$ is feasible (optimal) for $\mathrm{TGP}(T, T)$ as well. We proceed as follows.

(1) In Section 22.1 we assume that we are provided with some finite guard candidate set $G \subset T$ and show how to construct a witness set $W(G)$ with $|W(G)| \in \mathrm{O}(n|G|)$, such that any feasible solution of $\mathrm{TGP}(G, W(G))$ is feasible for $\mathrm{TGP}(G, T)$ as well.

(2) The main part is Section 22.2, where we propose a set of guard candidates $U$ with $|U| \in \mathrm{O}(n^2)$ and $\mathrm{OPT}(U, T) = \mathrm{OPT}(T, T)$.

(3) We combine the above steps in Section 22.3.

(4) With an efficient implementation in mind, we propose filtering techniques to reduce the size of the discretization in Section 22.4.

As detailed in Chapter 21, our discretization was developed independently from that of King [86] and is asymptotically smaller by a factor of $n$.

As a preliminary consideration, let us explore the extent to which the idea of *witnessable* polygons w.r.t. the AGP pursued by Chwa et al. [31] can be transferred to the TGP. Witnessable polygons allow placing a finite set of witnesses $W$, such that coverage of $W$ implies coverage of the entire polygon. The basic building blocks of Chwa et al. are *visibility kernels:* Given a point $w$ in a polygon, the visibility kernel of $w$ is the set of points that see at least as much as $w$ (definition for terrains below). Chwa et al. show

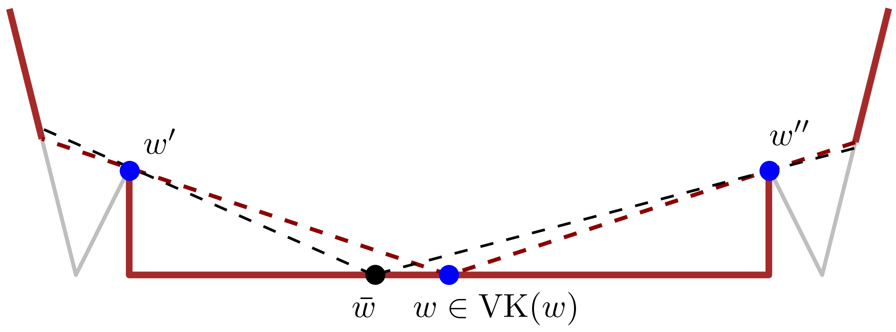


**Figure 22.1:** Witness $w$, $\mathcal{V}(w)$ highlighted in red, and its finite visibility kernel $\mathrm{VK}(w) = \{w, w', w''\}$ marked in blue. $\bar{w}$ has equivalent properties.

that a polygon admits a finite witness set if and only if it can be covered by a finite set of visibility kernels; this is not the case for arbitrary polygons.

This approach, however, does not carry over to the TGP. The visibility kernel of $w \in T$ is $\mathrm{VK}(w) := \{w' \in T \mid \mathcal{V}(w) \subseteq \mathcal{V}(w')\}$. Then for the terrain $T$ and $w \in T$ in Figure 22.1 we have $\mathrm{VK}(w) = \{w, w', w''\}$, so $\mathrm{VK}(w)$ is finite. The same argument holds for infinitely many $\bar{w} \in T$ near $w$. It follows that $T$ does not admit a finite visibility-kernel cover. Hence, applying the approach of Chwa et al. to $T$ does not yield a finite set of witness points, such that covering them implies coverage of $T$. Below, we choose a different approach and associate the witness set developed in Section 22.1 with a finite set of guard candidates instead of arbitrary guard locations.

## 22.1 Witnesses

Suppose we are given a terrain $T$, a finite set $G \subset T$ of guard candidates with $\mathcal{V}(G) = T$, and we want to cover $T$ using only guards $C \subseteq G$, i.e., we want to solve $\mathrm{TGP}(G, T)$. $G$ could be the set $V(T)$ of vertices to solve the VTGP or any other finite set, in particular, our guard candidates in Equation (22.5). We construct a finite set $W(G) \subset T$ of $\mathrm{O}(n|G|)$ witness points, such that all feasible solutions of $\mathrm{TGP}(G, W(G))$ are feasible for $\mathrm{TGP}(G, T)$.

Observe that any finite set of points on $T$ induces a subdivision of $T$. Due to the $x$-monotonicity of $T$ w.l.o.g. consider its projection on the $x$-axis, i.e., the closed interval spanned by the $x$-coordinates of $v_1$ and $v_n$, the leftmost and the rightmost vertex of $T$. This is a 1D arrangement[1] of vertices and open[2] line segments. Let $g \in G$ be a guard candidate. $\mathcal{V}(g)$ consists of $\mathrm{O}(n)$ subterrains, see Figure 21.1(a). The extremal points of $\mathcal{V}(g)$ induce an arrangement like above that we call the *visibility arrangement* $\mathcal{A}$ *of* $g$, where each open line segment and each vertex of $\mathcal{A}$ corresponds to a region on $T$ that is either entirely seen or not seen by $g$. We refer to the line segments of $\mathcal{A}$ as *visibility intervals* and to its vertices as *endpoints.* Next, consider the *overlay*[3] of all visibility arrangements of guards in $G$, i.e., the subdivision induced by all extremal visibility points of all guard candidates, see Figure 22.2(a). Every point in a *feature* $f$ (the region on $T$ corresponding to a visibility interval or endpoint) of the resulting visibility arrangement is entirely seen by the same set of guards

$$G(f) := \{g \in G \mid f \subseteq \mathcal{V}(g)\}\,. \tag{22.1}$$

**Observation 22.1.** *Let $f$ be a feature of the guard candidates' overlay and let $g \in G$ be a guard. Now consider an arbitrary witness $w \in f$. Then*

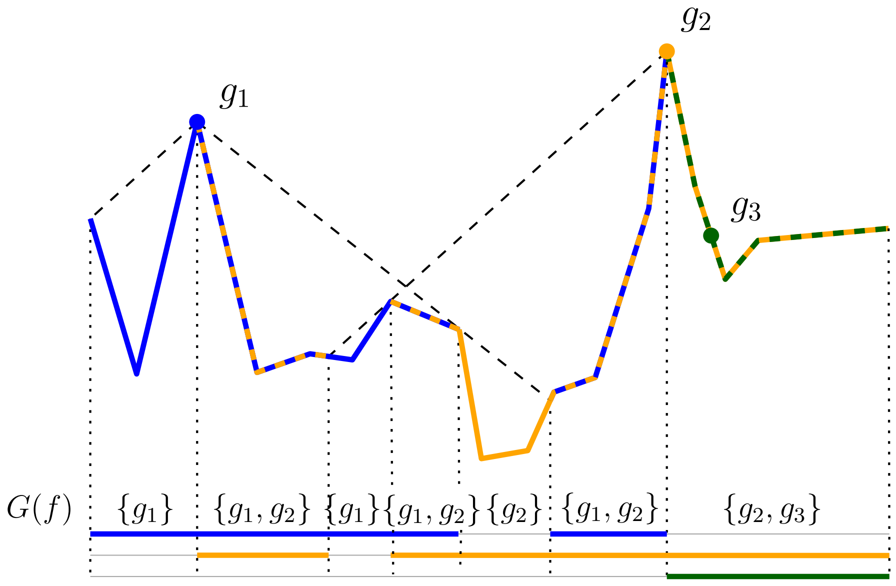
$$w \in \mathcal{V}(g) \quad \Leftrightarrow \quad f \subseteq \mathcal{V}(g) \quad \overset{(22.1)}{\Leftrightarrow} \quad g \in G(f). \tag{22.2}$$

Place one witness in each feature of the subdivision. Covering such a witness implies covering its entire feature, hence, covering all witnesses implies coverage of $T$. This

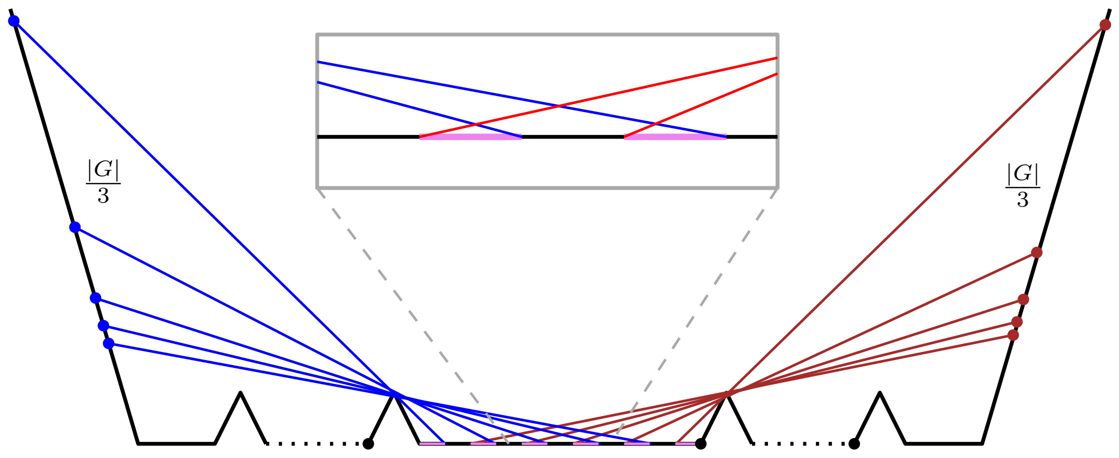
[1]Arrangements are a standard tool [2] that is, e.g., useful w.r.t. the AGP [40, 49].

[2]The leftmost and the rightmost line segment are not open when restricting the arrangement to $T$. This is irrelevant to the discussion at hand, however, including $v_1$ and $v_n$ into the point set w.l.o.g. ensures that all line segments are open.

[3]This is a standard technique in AGP-related problems [40, 49].



**(a)** Visibility overlay of $\mathcal{V}(g_1)$, $\mathcal{V}(g_2)$, and $\mathcal{V}(g_3)$ indicated in blue, orange, and green, respectively. Overlaps are indicated by altering colors.



**(b)** The set of inclusion-minimal features may have cardinality $\Theta(n|G|)$.

**Figure 22.2:** Witness discretization: visibility overlay (a) and a lower bound on the number of inclusion-minimal features (b).

requires $O(n|G|)$ witnesses, as each guard's visibility region has $O(n)$ extremal points which bounds the number of features by $O(n|G|)$. However, keeping efficient algorithms in mind, we reduce the number of witnesses, see Section 22.4.3: Similar to the shadow atomic visibility polygons by Couto et al. [36] — a successful strategy in AGP algorithms [40] — it suffices to include only those features $f$ with inclusion-minimal $G(f)$, i.e., those for which no $f'$ with $G(f') \subset G(f)$ exists:

**Theorem 22.2.** *Consider a terrain $T$ and a finite set of guard candidates $G$ with $\mathcal{V}(G) = T$. Let $F_G$ denote the set of features of the visibility overlay of $G$ on $T$ and $w_f \in f$ an arbitrary point in the feature $f \in F_G$. Then for*

$$W(G) := \{w_f \mid f \in F_G,\ G(f) \text{ is inclusion-minimal}\}, \tag{22.3}$$

*we have that if $C \subseteq G$ is feasible w.r.t.* $\text{TGP}(G, W(G))$*, then $C$ is also feasible w.r.t.* $\text{TGP}(G, T)$ *and*

$$\text{OPT}(G, W(G)) = \text{OPT}(G, T). \tag{22.4}$$

*Proof.* Let $C \subseteq G$ cover $W(G)$ and consider some point $w \in T$. We show that $w \in \mathcal{V}(C)$. By assumption, $w \in \mathcal{V}(G)$ and thus $w \in f$ for some feature $f \in F_G$. The set $W(G)$ contains some witness in $w_f \in f$ or a witness $w_{f'} \in f'$ with $G(f') \subseteq G(f)$ by construction. In the first case, $w$ must be covered, otherwise $w_f$ would not be covered and $C$ would be infeasible for $\text{TGP}(G, W(G))$. In the second case $w_{f'}$ is covered, so some guard in $G(f')$ is part of $C$, and that guard also covers $f$ and therefore $w$.

Regarding Equation (22.4), observe that $\text{TGP}(G, W(G))$ is a relaxation of $\text{TGP}(G, T)$, so $\text{OPT}(G, W(G)) \leq \text{OPT}(G, T)$ follows. Furthermore, if $C$ is feasible and optimal w.r.t. $\text{TGP}(G, W(G))$, it is also feasible for $\text{TGP}(G, T)$ as argued above. It follows that $|C| = \text{OPT}(G, W(G)) \geq \text{OPT}(G, T)$, proving Equation (22.4). □

**Observation 22.3.** *Using the set of one witness per inclusion-minimal feature as in Equation* (22.3) *does not reduce the worst-case complexity of* $|W(G)| \in O(n|G|)$ *witnesses.*

*Proof.* See Figure 22.2(b). For $|G| \in \Theta(n)$ consider the terrain with $\Theta(n)$ valleys with $\frac{|G|}{3}$ guards placed on the left (blue) and the right (red) slope each. In addition there is one guard (black) placed in each valley. Thus, each of the $\Theta(n)$ valleys contains $\Theta(|G|)$ inclusion-minimal intervals depicted in violet, resulting in $\Theta(n|G|)$ inclusion-minimal features. □

Nevertheless, using only inclusion-minimal witnesses significantly speeds up our implementation, refer to Section 22.4.3 and Chapter 24.

**Observation 22.4.** *Let $p$ be an endpoint of the visibility overlay of $G$ on $T$. Then $W(G)$ does not require a witness corresponding to $p$: $p$ is incident to at least one visibility interval $I$ and we have $G(p) \supseteq G(I)$ because visibility regions are closed sets. Hence, $G(p)$ is not inclusion-minimal.*

## 22.2 Guards

Throughout this section, let $T$ be a terrain, $V = V(T)$ its vertices, and $E = E(T)$ its edges. Let $C \subset T$ be a finite guard cover of $T$ that is feasible w.r.t. $\text{TGP}(T, T)$ and possibly optimal. Define $U$ as all vertices along with the extremal points of their visibility regions:

$$U := V \cup \bigcup_{v \in V} \{p \mid p \text{ is extremal in } \mathcal{V}(v)\}\,. \tag{22.5}$$

**Observation 22.5.** *$|U| \in \mathrm{O}(n^2)$ as noted by Ben-Moshe et al. [18]: It contains $n$ vertices, each associated with $\mathrm{O}(n)$ subterrains of at most two extremal points.*

Ben-Moshe et al. use a similar set, but they also add an arbitrary point of $T$ between each pair of consecutive points in $U$. They need these points as witnesses. We, however, keep the witnesses separate by our definition of $\mathrm{TGP}(G, W)$.

In the remainder of this section we show that $U$ contains all guard candidates necessary for solving the CTGP, $\mathrm{TGP}(T, T)$, i.e., that $\mathrm{OPT}(U, T) = \mathrm{OPT}(T, T)$. Recall that $\mathrm{OPT}(T, T) \leq \frac{n}{2}$. Our strategy is to show that in any cover $C$ of $T$ it is always possible to move a guard in $C \setminus U$ to a carefully chosen point in $U$ without losing coverage. This procedure preserves the cardinality and feasibility of any feasible cover; iterating it results in a cover $C \subseteq U$. In particular, this is possible for an optimal guard cover.

First observe that an edge that is entirely covered by a guard $g \in C \setminus U$ is still covered after moving $g$ to one of its neighbors in $U$.

**Lemma 22.6.** *Let $g \in C \setminus U$ be a guard that covers an entire edge $e_i \in E$, i.e., $e_i \subseteq \mathcal{V}(g)$. Then $u_\ell$ and $u_r$, the $U$-*neighbors *of $g$, with*

$$u_\ell := \max\{u \in U \mid u < g\} \tag{22.6}$$
$$u_r := \min\{u \in U \mid g < u\} \tag{22.7}$$

*each entirely cover $e_i$, too, i.e., $e_i \subseteq \mathcal{V}(u_\ell)$ and $e_i \subseteq \mathcal{V}(u_r)$.*

*Proof.* $g$ covers $e_i$, so $v_i, v_{i+1} \in \mathcal{V}(g)$, implying $g \in \mathcal{V}(v_i) \cap \mathcal{V}(v_{i+1})$. Moving $g$ towards $u_\ell$ does not move $g$ out of $\mathcal{V}(v_i)$ or $\mathcal{V}(v_{i+1})$, as the boundaries of those regions are contained in $U$ by construction. Hence, $v_i, v_{i+1} \in \mathcal{V}(u_\ell)$ and thus $e_i \subseteq \mathcal{V}(u_\ell)$. Analogously, $e_i \subseteq \mathcal{V}(u_r)$. □

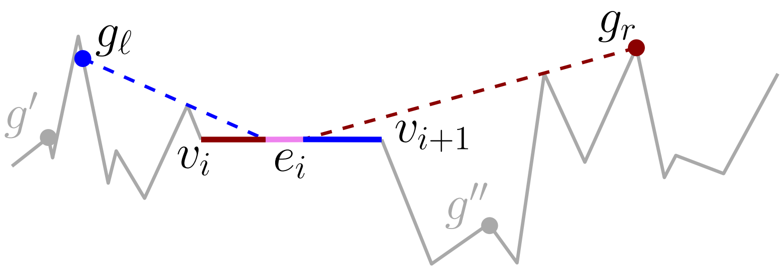
It remains to consider the edges not entirely covered by a single guard, refer to Figure 22.3(a). We refer to such edges as *critical edges:*

**Definition 22.7** (Critical Edge). *An edge $e \in E$ is* critical w.r.t. *$g \in C$ if $C \setminus \{g\}$ covers some part of, but not all of,* $\mathrm{int}(e)$*. If $e$ is critical w.r.t. some $g \in C$ we call $e$ a* critical edge.

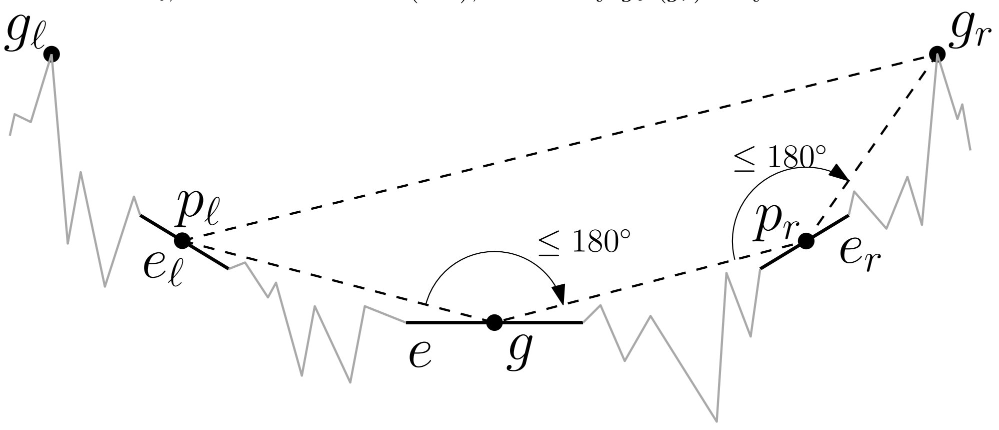
So $e$ is critical if and only if more than one guard is responsible for covering $\mathrm{int}(e)$.

**Definition 22.8** (Left-Guard/Right-Guard). *$g \in C$ is a* left-guard (right-guard) *of $e_i \in E$ if $g < v_i$ ($v_{i+1} < g$) and $e_i$ is critical w.r.t. $g$. We call $g$ a* left-guard (right-guard) *if it is a left-guard (right-guard) of some $e \in E$.*

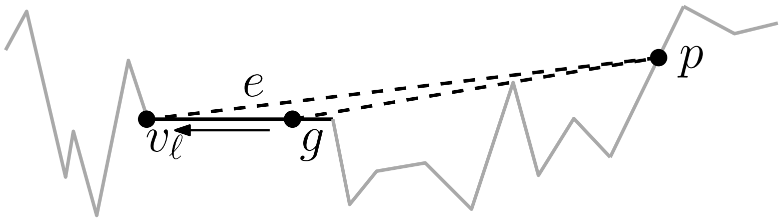
For the sake of completeness, we state and prove the following lemma which also follows from the well-established order claim [18]:



**(a)** The edge $e_i$ is critical w.r.t. $g_\ell$ and $g_r$: The right (left) part of $e_i$, indicated in blue (red), is seen by $g_\ell$ ($g_r$) only.



**(b)** No guard $g$ is both left- and right-guard. Any point on the critical edge $e_\ell$ seen by $g$ is also seen by $g_r$, hence $e_\ell$ cannot be critical w.r.t. $g$.



**(c)** Moving the left-guard $g$ to the left. Any point $p$ that $g$ sees to its right remains visible while moving $g$ towards $v_\ell$.

**Figure 22.3:** Guard discretization: critical edges (a) and dominated guards (b)–(c).

**Lemma 22.9.** *Let $g \in C$ be a guard left of $v_i$ (right of $v_{i+1}$) such that $g$ covers a non-empty subset of* $\operatorname{int}(e_i)$*. Then $g$ covers a single interval of $e_i$, including $v_{i+1}$ (including $v_i$). In particular, this holds if $g$ is a left-guard (right-guard) of $e_i$.*

*Proof.* Refer to Figure 22.3(a), where $g_\ell$ ($g_r$) depicts the guard left of $v_i$ (right of $v_{i+1}$). Obviously, $g_\ell$ is nowhere below the line supporting $e_i$. Let $p$ be a point on $e_i$ seen by $g_\ell$. It follows that $\overline{g_\ell p}$ and $\overline{pv_{i+1}}$ form an $x$-monotone convex chain that is nowhere below $T$. Thus, $\overline{g_\ell v_{i+1}}$ is nowhere below $T$. It follows that $g_\ell$ sees $v_{i+1}$ and any point on $\overline{pv_{i+1}}$. A symmetric argument holds for $g_r$ and $v_i$. □

**Corollary 22.10.** *For a critical edge there is exactly one left- and exactly one right-guard.*

*Proof.* Suppose for the sake of contradiction that $g, g' \in C$ are distinct critical left-guards of $e_i \in E$. By Lemma 22.9, $I := \mathcal{V}(g) \cap e_i$ and $I' := \mathcal{V}(g') \cap e_i$ are connected intervals on $e_i$ with $v_{i+1} \in I, I'$. Assume w.l.o.g. that $I' \subseteq I$. This contradicts $g'$ being a left-guard of $e_i$ because $g$ dominates $g'$ on $e_i$, i.e., $e_i \subseteq \mathcal{V}(C \setminus \{g'\})$. So $e_i$ has exactly one critical left-guard. A symmetric argument shows that $e_i$ has exactly one right-guard. □

**Corollary 22.11.** *Let $e_i \in E$ be a critical edge and let $g_\ell, g_r \in C$ be its left- and right-guards. Then $\mathcal{V}(g_\ell) \cap e_i \cap \mathcal{V}(g_r) \neq \emptyset$.*

*Proof.* Consider $I_\ell := e_i \cap \mathcal{V}(g_\ell)$. By Lemma 22.9, $I_\ell$ is a single interval on the right of $e_i$, i.e., from some point $p_\ell \in \operatorname{int}(e_i)$ up to its right vertex $v_{i+1}$. Since $e_i$ and $\mathcal{V}(g_\ell)$ are closed sets, so is $I_\ell$, hence, $p_\ell$ is well-defined. Analogously, define $I_r := e_i \cap \mathcal{V}(g_r)$ and let $p_r$ be the rightmost point in $I_r$. Let $p_m := \frac{1}{2}(p_\ell + p_r) \in e_i$ be the midpoint of the interval boundaries. We argue in two steps: (1) Observe that $I_\ell \cup I_r = e_i$. Otherwise, $p_m \notin I_\ell, I_r$ would be covered by some guard $g \in C \setminus \{g_\ell, g_r\}$. By Lemma 22.9, $g$ would dominate either $g_\ell$ or $g_r$ on $e_i$, contradicting the precondition that $g_\ell$ and $g_r$ are critical w.r.t. $e_i$. (2) Due to $e_i = I_\ell \cup I_r$, we have $p_m \in I_\ell \cap I_r$. Hence, $p_m \in \mathcal{V}(g_\ell) \cap e_i \cap \mathcal{V}(g_r)$ and the claim follows. □

By Lemma 22.6, we can move non-critical guards to one of their neighbors in $U$ because they are only responsible for entire edges. Unfortunately, this is impossible if $g \in C \setminus U$ is a left- or a right-guard: We might lose coverage of some part of an edge that is critical w.r.t. $g$. However, the following lemma establishes that we can move a left-guard $g$ to its left neighbor vertex if $g$ is not a right-guard (a symmetric version for non-left-guards follows).

**Lemma 22.12.** *Let $C$ be some finite cover of $T$, let $g \in C \setminus V$ be a left- but not a right-guard, and let $v_\ell := \max\{v \in V \mid v < g\}$ be the rightmost vertex left of $g$. Then*

$$C' = (C \setminus \{g\}) \cup \{v_\ell\} \tag{22.8}$$

*is a guard cover of $T$.*

*Proof.* Since $g$ is a left-guard of some critical edge $e_r$, there is a corresponding right-guard $g_r$ of $e_r$ by Corollary 22.10, see Figure 22.3(b). Let $p_\ell \in \{p \in \mathcal{V}(g) \mid p \leq g\}$ be a point that $g$ sees to its left. We show that $p_\ell$ is seen by $g_r$: Consider $p_r$, a point in

$\mathcal{V}(g) \cap e_r \cap \mathcal{V}(g_r)$, which exists by Corollary 22.11. $\overline{p_\ell g}$, $\overline{g p_r}$, and $\overline{p_r g_r}$ form a convex chain (convex due to $g, p_r \notin V$) that is nowhere below $T$, so $p_\ell \in \mathcal{V}(g_r)$. Thus, $g$ is dominated to its left by $g_r$. Moreover, $g$ is dominated to its right by $v_\ell$, see Figure 22.3(c): Let $p \in \{p \in \mathcal{V}(g) \mid g \leq p\}$ be a point seen by $g$ located to its right. Then $\overline{v_\ell g}$ and $\overline{gp}$ form a convex chain nowhere below $T$, so $p \in \mathcal{V}(v_\ell)$. In conclusion, replacing $g$ by $v_\ell$ in $C$ yields a feasible cover because $\{p \in \mathcal{V}(g) \mid p \leq g\}$ is covered by $g_r$ and $\{p \in \mathcal{V}(g) \mid g \leq p\}$ by $v_\ell$. □

**Corollary 22.13.** *Let $C$ be some finite cover of $T$, let $g \in C \setminus V$ be a right- but no left-guard, and let $v_r := \min\{v \in V \mid g < v\}$ be the leftmost vertex right of $g$. Then*

$$C' = (C \setminus \{g\}) \cup \{v_r\} \tag{22.9}$$

*is a guard cover of $T$.*

So far, the status is that guards in $C \setminus U$ that are neither left- nor right-guard can be moved to a $U$-neighbor. Left-guards (right-guards) that are no right-guard (left-guard) can be moved to the next vertex to the left (right). The remaining case, i.e., guards that are both left- and right-guards, cannot happen:

**Lemma 22.14.** *Let $C$ be a finite cover of $T$. No $g \in C \setminus V$ is both a left- and a right-guard.*

*Proof.* Refer to Figure 22.3(b). Suppose for the sake of contradiction that $g \in C \setminus V$ is the left-guard of an edge $e_r$ ($e_r$ is to the right of $g$) and the right-guard of edge $e_\ell$ ($e_\ell$ is to the left of $g$). Since $e_r$ is critical, there must be a right-guard $g_r$ of $e_r$ by Corollary 22.10. By Corollary 22.11 there is a point $p_r \in e_r$ seen by $g$ and $g_r$. As $g \notin V$, $g \in \operatorname{int}(e)$ for some edge $e$.

Now consider some point $p_\ell \in \mathcal{V}(g)$ such that $p_\ell < g$. $p_\ell$ and $p_r$ are not below the line supported by $e$ and the same holds for $g$ and $g_r$ w.r.t. $e_r$. It follows that segments $\overline{p_\ell g}$, $\overline{g p_r}$, and $\overline{p_r g_r}$ form an $x$-monotone convex chain that is nowhere below $T$. Hence, $p_\ell \in \mathcal{V}(g_r)$. Since $p_\ell$ was arbitrary, any point $p \in \mathcal{V}(g)$ to the left of $g$ is also seen by $g_r$, a contradiction to $g$ being a right-guard. □

The next theorem shows that the set $U$ as defined in Equation (22.5) contains all guard candidates necessary for a minimum-cardinality guard cover of $T$.

**Theorem 22.15.** *Let $T$ be a terrain and consider $U$ from Equation* (22.5)*. Then we have*

$$\mathrm{OPT}(U, T) = \mathrm{OPT}(T, T). \tag{22.10}$$

*Proof.* Let $C$ be optimal w.r.t. $\mathrm{TGP}(T, T)$ and observe that $C$ is finite, as placing a guard on every other vertex is feasible. We show how to replace a single guard $g \in C \setminus U$ by one in $U$ while maintaining feasibility, i.e., $\mathcal{V}(C) = T$. The claim then follows by induction.

Should $g$ be neither left- nor right-guard, it can be replaced by a neighboring point in $U$ by Lemma 22.6. If $g$ is a left-, but not a right-guard (or vice versa), it can be replaced by its left (right) neighbor in $V \subseteq U$ by Lemma 22.12 (Corollary 22.13). Lemma 22.14 asserts that $g$ cannot be a left- and a right-guard at the same time. □

## 22.3 Full Discretization

We formulate the key result of this chapter: The CTGP, i.e., finding a minimum-cardinality guard cover $C$ guarding an entire terrain $T$, without any restriction on where on $T$ the guards can be placed, is a discrete problem with a discretization $(U, W(U))$ of size $\mathrm{O}(n^3)$ and polynomially bounded coordinate complexity.

**Theorem 22.16.** *Let $T$ be a terrain, and consider $U$ and $W(U)$ from Equations* (22.5) *and* (22.3)*. Then the following claims hold.*

*(1) If $C \subseteq U$ is feasible w.r.t.* $\mathrm{TGP}(U, W(U))$ *then $C$ is feasible w.r.t.* $\mathrm{TGP}(T, T)$*. Furthermore,*

$$\mathrm{OPT}(T, T) = \mathrm{OPT}(U, W(U)). \tag{22.11}$$

*(2) We have $|U| \in \mathrm{O}(n^2)$ and $|W(U)| \in \mathrm{O}(n^3)$.*

*(3) Let $B$ be the largest number of bits required to represent a coordinate of $V$. The number of bits required to represent the coordinates of a guard candidate $g \in U$ is polynomial in $B$.*

*Proof.* To show (1), observe that if $C$ is feasible w.r.t. $\mathrm{TGP}(U, W(U))$, it is feasible w.r.t. $\mathrm{TGP}(U, T)$ by Theorem 22.2; feasibility of $C$ w.r.t. $\mathrm{TGP}(T, T)$ follows from $U \subset T$. Equation (22.11) holds due to

$$\mathrm{OPT}(T, T) \overset{(22.10)}{=} \mathrm{OPT}(U, T) \overset{(22.4)}{=} \mathrm{OPT}(U, W(U)). \tag{22.12}$$

Claim (2) follows from Observations 22.5 and 22.3. Regarding (3), observe that if $g \in V$, the claim trivially holds. Otherwise, $g \in U \setminus V$ is the intersection of two lines, each spanned by two vertices in $V$. □

## 22.4 Reducing the Size of the Discretization

While $\mathrm{O}(n^2)$ guard candidates and $\mathrm{O}(n^3)$ witnesses, see Theorem 22.16, may be sufficient from a theoretical point of view, it is imperative to reduce their numbers for an efficient implementation. We propose filtering techniques that, while not reducing the asymptotic size of the discretization, typically remove around 90 % of the guard candidates and an even larger fraction of the witnesses. The experiments in Chapter 24 demonstrate this to be a key success factor that increases the size of solvable instances by several orders of magnitude.

We say that $g \in T$ *dominates* $g' \in T$ if $\mathcal{V}(g') \subseteq \mathcal{V}(g)$, in which case $g'$ can be safely discarded. A core issue, however, is that visibility calculations are expensive, so the key challenge is to identify dominated guard candidates *without determining their visibility region.* The guard filter in Section 22.4.2 is capable of that. We propose guard filters in Sections 22.4.1 and 22.4.2, discuss a witness filter in Section 22.4.3, and present an open problem in Section 22.4.4.

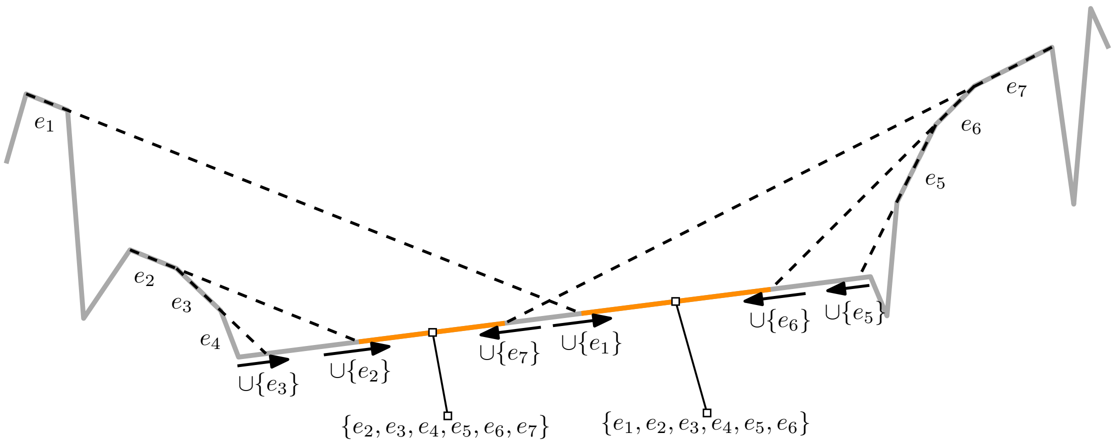


**Figure 22.4:** Edge-interior guards are only responsible for entire edges. Edges can only become visible when crossing some $u \in U$, the arrows indicate in which direction. Only the orange regions contain guard candidates that are inclusion-maximal w.r.t. entire edges.

### 22.4.1 Filtering Dominated Guards

Let $T$ be a terrain and $G \subset T$ a finite set of guard candidates with $\mathcal{V}(G) = T$. Consider $g, g' \in G$, suppose we know $\mathcal{V}(g)$ and $\mathcal{V}(g')$, and observe that checking whether $g$ dominates $g'$ takes $O(n)$ time since visibility regions consist of $O(n)$ subterrains. Moreover, removing all dominated guards from $G$ requires $O(|G|^2)$ domination queries, i.e., an intolerable $O(n^5)$ time when applied to $G = U$ from Equation (22.5).

Instead, we propose a heuristic using $O(|G|)$ domination queries and thus an acceptable $O(n^3)$ time for $G = U$. Suppose $G$ is ordered w.r.t. $x$-coordinates. The *local domination filter* removes all guard candidates that are dominated by one of their neighbors. This is based on the observation that neighboring guards' visibility regions often are quite similar or one clearly dominates the other (a local "dent"). Experiments demonstrate that this strategy is beneficial in terms of time and memory consumption, see Chapter 24.

### 22.4.2 Filtering Edge-Interior Guards

Let $T$ be a terrain, $U$ the guard candidates from Equation (22.5), and fix an edge $e$. By Lemma 22.12, Corollary 22.13, and Lemma 22.14 assume w.l.o.g. that all critical guards are located at the vertices. Hence, guards in $U_e := U \cap \operatorname{int}(e)$ are only responsible for covering entire edges. Recall that by construction of $U$, when moving across $u \in U_e$, a vertex becomes visible or invisible, depending on the direction. Furthermore, observe that covering an entire edge is equivalent to seeing both its vertices. The sets of edges entirely seen by each $u \in U_e$,

$$E_u := \{e \in E \mid e \subseteq \mathcal{V}(u)\} = \{e_i \in E \mid v_i, v_{i+1} \in \mathcal{V}(u)\}, \tag{22.13}$$

define a partial order on $U_e$ w.r.t. inclusion as indicated in Figure 22.4. Most importantly, $u$ is inclusion-maximal if $E_{u'} \not\supset E_u$ for all $u' \in U_e$. We show that it suffices to consider the guard candidates that are inclusion-maximal w.r.t. $E_u$:

**Theorem 22.17.** *Let $U'_e \subseteq U_e$ be the set that only contains inclusion-maximal guard candidates w.r.t. entire edges, as defined above. Then*

$$U' := (U \setminus U_e) \cup U'_e \tag{22.14}$$

*admits covering $T$ with the same number of guards as $U$, i.e.,*

$$\mathrm{OPT}(U', T) = \mathrm{OPT}(U, T). \tag{22.15}$$

*Proof.* No guard can be a left- and a right-guard at the same time by Lemma 22.14. Furthermore, by Lemma 22.12 (Corollary 22.13), a left-guard (right-guard) can be moved to its left (right) neighbor in $V$. Thus, w.l.o.g., $u \in U_e$ is no left- or right-guard, because $U_e$ does not contain vertices by definition. Hence, no edge is critical w.r.t. $u$ by Definition 22.7; $u$ is only responsible for covering entire edges and can be replaced by its inclusion-maximal sibling in $U'_e$ without changing the feasibility or cardinality of a cover of $T$. □

The key is that identifying guard candidates $u \in U \setminus V$ that are not inclusion-maximal w.r.t. entire edges can be implemented without determining $\mathcal{V}(u)$: For each $u \in U \setminus V$, store a reference to the vertex whose visibility region is extremal at $u$, as well as whether it is situated to the left or to the right of $u$. This allows to decide which vertex becomes visible or invisible when sweeping across $u$ from left to right, as indicated in Figure 22.4.

We use the following sweep-line algorithm. For every $e \in E$, sweep through $U_e$ from left to right. While encountering $u \in U_e$ where new vertices become visible, do nothing. When reaching the first $u \in U_e$ where a vertex becomes invisible, report $u$. Since $u$ is inclusion-maximal w.r.t. vertices, it is inclusion-maximal w.r.t. entire edges. Then ignore all points corresponding to vertices becoming invisible until encountering the first that becomes visible and continue as above. This algorithm reports guard candidates that are inclusion-maximal w.r.t. vertices. However, no vertex can become visible and invisible in two distinct $u, u' \in U_e \subset \mathrm{int}(e)$ by Lemma 22.9, hence the reported guard candidates are inclusion-maximal w.r.t. edges, as claimed.

This efficiently discards up to 98 % of the guard candidates in the CTGP — without determining their visibility regions — and essentially removes the computational boundary between VTGP and CTGP as demonstrated in Chapter 24.

**Observation 22.18.** *The above sweep-line algorithm needs only the visibility regions of vertices, not those of $U \setminus V$. Deciding whether a vertex $v$ becomes visible or invisible at $u \in U_e$ depends only on whether $u$ is extremal in $\mathcal{V}(v)$ and on $v < u$, as described above.*

**Observation 22.19.** *Filtering $U$ as above still may yield $\Theta(n^2)$ guard candidates: Insert a vertex below each guard on the slopes in Figure 22.2(b). Then every other interval is inclusion-maximal w.r.t. the vertices on the slopes.*

### 22.4.3 Filtering Witnesses

Let $U$ be a possibly filtered set of guard candidates. The construction of the witness set $W(U)$ as in Equation (22.3) already includes a filtering mechanism: Only inclusion-minimal witnesses need to be kept. Observe that a smaller, filtered, $U$ automatically yields a smaller $W(U)$. Furthermore, observe that in terms of an implementation witnesses are much cheaper then guard candidates: They require no visibility region or coordinates — by Observation 22.1, they only need to store references to the guards covering them.

We acquire witnesses very much like guards in Section 22.4.2: Sort the extremal points of all guard candidates' visibility regions by their $x$-coordinates. For each of these points we know whether a visibility region opens or closes and to which guard it is associated. Sweeping through these points, it is straightforward to keep track of which guard candidates see the current event point and where this set is inclusion-minimal.

Our approach keeps witnesses that are locally, but not necessarily globally, inclusion-minimal. We can efficiently exploit the underlying geometry to identify the locally inclusion-minimal witnesses, but it is an open question whether globally inclusion-minimal witnesses can be identified just as efficiently. However, experiments demonstrate that our approach is extremely effective, see Chapter 24.

### 22.4.4 Open Problem

Our discretization has size $|U| + |W(U)| \in \mathrm{O}(n^3)$ by Theorem 22.16. The filters prove invaluable to our implementation by quickly reducing, on average, the size by more than 90 % — see Chapter 24 — but do not reduce the asymptotic complexity. In the interest of finding a small discretization, observe that a discretization that minimizes $|G| + |W|$ is one where $|G| = \mathrm{OPT}(T, T)$, i.e., just as hard to find as solving $\mathrm{TGP}(T, T)$. Is there a discretization, obtainable in polynomial time, of size $\mathrm{o}(n^3)$?

# CHAPTER 23

# NP-Completeness and Approximation

The results in this chapter directly follow from the existence of a polynomially-sized discretization of the CTGP — see Chapter 22 — and are a key motivation for its development. As as detailed in Chapter 21, our discretization was developed independently from but after that of King [86], hence, the results in this chapter are implicit to prior work. Nevertheless, we present them as they are key consequences of Chapter 22.

For a long time, the NP-hardness of the decision variants of the CTGP and the VTGP were generally assumed but not shown until 2010 by King and Krohn [87].[1] The NP-completeness of the decision variant of the VTGP immediately follows. In Theorem 23.1, we establish that the decision version of the CTGP is also a member of NP, and thus NP-complete, which is possible because polynomially many bits suffice to represent all required guard coordinates. This is surprising as the more general AGP has been recently shown to require irrational guard coordinates [1], even in monotone polygons.

**Theorem 23.1.** *The decision variant of the Continuous Terrain Guarding Problem (CTGP), i.e., the following decision problem is NP-complete:*

**Input** *A terrain $T$ with rational vertices $V(T) \subset \mathbb{Q}^2$ and $k \in \mathbb{N}$.*

**Output** Yes *if the CTGP permits $k$ guards $G = \{g_1, \dots, g_k\} \subset T$ with $\mathcal{V}(G) = T$, i.e., if $\mathrm{OPT}(T,T) \leq k$, and* No *otherwise.*

*Proof.* The above problem is known to be NP-hard [87]. It remains to show that it is a member of NP: A non-deterministic Turing machine computes $U$ from Equation (22.5), which is possible in polynomial time by Theorem 22.16, and guesses guards $C \subseteq U$ with $|C| \leq k$. It then verifies $\mathcal{V}(C) = T$ in polynomial time [75]. □

Our discretization can be combined with the PTAS for discrete TGP variants by Gibson et al. [66].

**Theorem 23.2.** *There exists a PTAS for the Continuous Terrain Guarding Problem (CTGP). That is, for any constant $\varepsilon > 0$, there is a polynomial-time algorithm which, given a terrain $T$, returns $C \subset T$ with $\mathcal{V}(C) = T$ and $|C| \leq (1+\varepsilon)\,\mathrm{OPT}(T,T)$.*

*Proof.* Using Equations (22.5) and (22.3), we determine the sets $U$ and $W(U)$ for $T$ with $|U| + |W(U)| \in \mathrm{O}(n^3)$ in polynomial time by Theorem 22.16. Using the PTAS for $\mathrm{TGP}(G,W)$ with finite $G, W \subset T$ by Gibson et al. [66], we compute $C \subseteq U \subset T$ with

$$|C| \overset{[66]}{\leq} (1+\varepsilon)\,\mathrm{OPT}(U, W(U)) \overset{(22.11)}{=} (1+\varepsilon)\,\mathrm{OPT}(T,T) \tag{23.1}$$

in polynomial time, where $C$ is feasible w.r.t. $\mathrm{TGP}(T,T)$ by Theorem 22.16. □

[1]The conference version of King and Krohn [87] appeared 2010.

# CHAPTER 24

# Solver

The discretization and the filters proposed in Chapter 22 allow for an efficient implementation of a solver for the TGP: First determine the guard locations (or start with user-provided guard candidates when solving a discrete TGP variant), apply the filters, and generate the witnesses. At this point, the problem is a finite instance of SC which can be solved by an external solver; we use an IP solver. The IP is presented in Section 24.1. We propose the algorithm in Section 24.2. It solves instances of the TGP with up to $10^6$ vertices within roughly 1–2 minutes on a standard desktop computer, as demonstrated in Section 24.3.

Our approach is to efficiently determine and filter a discretization; once the discretization is obtained, the resulting SC instance is solved by an IP solver. Focusing on the polynomial-time part of the algorithm and delegating its NP-hard part is founded in our experience with the closely related AGP. It has been established that in many AGP-solver variants, the geometric subroutines and not the SC instances form the bottleneck in terms of real-world performance [40]. Finding a discretization and the overhead of the EGC paradigm, i.e., the use of exact number types, exhaust computational resources long before the SC instances become large enough to be problematic. Furthermore, SC is a well-studied problem and state-of-the-art solvers benefit from that.

We admit that instances deliberately designed to be hard for the underlying IP solver are not covered by the above line of reasoning. Still, we choose to focus on an efficient reduction of TGP to SC instances, as this is required by any fast solver for the TGP. As a consequence, we evaluate our algorithm's performance in the presence and absence of the individual filtering techniques. It turns out that these techniques are critical for good performance.

## 24.1 IP Formulation

Let $T$ be a terrain and let $G, W \subset T$ be finite sets of guard candidates and witnesses such that $W \subseteq \mathcal{V}(G)$. In that case, $\text{TGP}(G, W)$ is an instance of SC:[1] We ask for a cover of the set $W$ by a minimal number of subsets from $\{\mathcal{V}(g) \cap W \mid g \in G\}$. Hence, we obtain the following IP formulation of $\text{TGP}(G, W)$:

$$\min \sum_{g \in G} x_g \tag{24.1}$$

$$\text{s.t.} \sum_{g \in \mathcal{V}(w) \cap G} x_g \geq 1 \quad \forall w \in W \tag{24.2}$$

$$x_g \in \{0, 1\} \qquad \forall g \in G. \tag{24.3}$$

[1]Refer to, e.g., Garey and Johnson [63] for a definition of SC.

```
input: Terrain $T$ and mode $\in$ {VertexGuards, PointGuards}   ▷ VTGP or CTGP
output: Guard cover of $T$
1: $U \leftarrow V(T)$                       ▷ vertices are guard candidates in both modes
2: $W \leftarrow \emptyset$
3: for each $u \in U$ do
4:     determine $\mathcal{V}(u)$
5: end for

6: if PointGuards then                           ▷ the alternative is VertexGuards
7:     $U \leftarrow U \cup \bigcup_{v \in U} \{p \mid p \text{ is extremal in } \mathcal{V}(v)\}$      ▷ Equation (22.5)
8:     if EdgeFilter then
9:         filter edge-interior guards in $U$ by sweep-line algorithm      ▷ Section 22.4.2
10:    end if
11:    for each $u \in U \setminus V(T)$ do
12:        determine $\mathcal{V}(u)$                 ▷ after EdgeFilter, see Section 22.4.2
13:    end for
14: end if

15: if DominationFilter then
16:     filter out guards in $U$ dominated by a neighbor              ▷ Section 22.4.1
17: end if

18: if WitnessFilter then
19:     $W \leftarrow$ inclusion-minimal features of the overlay of $U$          ▷ Equation (22.3)
20: else
21:     $W \leftarrow$ all features from the overlay of $U$              ▷ unfiltered Equation (22.3)
22: end if

23: solve TGP($U, W$) with an IP solver                      ▷ Equations (24.1)–(24.3)
```

**Algorithm 24.1:** Optimal TGP solutions.

A binary variable $x_g$ for each guard candidate $g \in G$ indicates whether $g$ is picked: $x_g = 1$ if and only if $g$ is part of the cover. For each witness $w \in W$, a constraint ensures that $w$ is covered by at least one guard. We minimize the number of guards in the cover. This is a standard IP formulation that is, e.g., used in AGP solvers [40].

Choosing $G = U$ (possibly filtered) and $W = W(U)$ from Equations (22.5) and (22.3), (24.1)–(24.3) model the CTGP. In order to solve the VTGP, pick $G = V$ and $W = W(V)$. For an arbitrary a priori guard discretization $G$, i.e., for TGP($G, T$), pick $W = W(G)$.

## 24.2 Algorithm

Algorithm 24.1 has two modes.

(1) PointGuards solves TGP($T, T$) and

(2) VertexGuards solves TGP($V, T$).

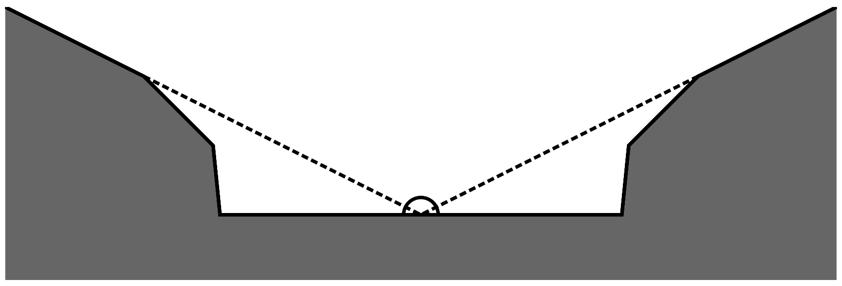


**Figure 24.1:** Exact computations and the Terrain Guarding Problem: The guard candidate marked by the white circle lies exactly on the intersection of three lines spanned by edges of the terrain. If it is not found due to rounding errors the solution is suboptimal.

Everything except lines 6–14 applies to both modes; lines 6–14 generate non-vertex guard candidates and possibly filter them. Filtering mechanisms are activated individually:

(1) DominationFilter from Section 22.4.1 removes guard candidates that are dominated by one of their neighbors,

(2) EdgeFilter corresponds to the guard filter from Section 22.4.2 and is only applicable in the PointGuards mode, and

(3) WitnessFilter determines whether only the inclusion-minimal witness are used, refer to Equation (22.3) and Section 22.4.3.

We remark two things about line 23. (1) It is the only subroutine that may require superpolynomial time, due to the NP-hardness of the TGP [87]. In our experiments, however, this is not the bottleneck of our algorithm; the geometric subroutines require most time and memory. We discuss this in Section 24.3.4. (2) Algorithm 24.1 transforms an instance of VTGP or CTGP into an instance of SC which it delegates to a solver. An IP or SAT solver, an SC approximation algorithm, the PTAS by Gibson et al. [66], or any other solver (including those oblivious to the underlying geometry) can be used as a drop-in replacement. Observe that all solvers benefit from our filtering framework. However, since benchmarking the underlying solver is not our concern, we restrict our experiments to a state-of-the-art IP solver.

## 24.2.1 Implementation

We implemented Algorithm 24.1 in C++11 and compiled with g++-4.8.4 [64]. The geometric subroutines use CGAL-4.6 [28] (the Computational Geometry Algorithms Library) with the visibility implementation of Haas and Hemmer [70] and we solve IPs using CPLEX-12.6.0 [37]. Furthermore, we use boost-1.58.0 [23] and simple-svg-1.0.0 [126].

Like CGAL, our implementation adheres to the EGC paradigm, compare, e.g., Pion and Fabri [121]. In a nutshell, this means that no geometric subroutine introduces rounding errors. As an illustration of why this is necessary consider the terrain $T$ in Figure 24.1.[2] There is exactly one optimal solution of TGP($T, T$) and it requires the

[2]We use $T$ as a test case of our software.

guard $g$ at the intersection of three lines: the lines spanned by the leftmost, the rightmost, and the bottom edge. If the coordinates of $g$ are only slightly off, for example due to the use of floating-point arithmetic, an optimal solution is not found. Similarly, rounding errors may introduce infeasibilities as the guards are not located on the terrain or — due to imprecision in the calculated visibility regions — do not cover all of it. The core of the problem is that even slightly perturbed coordinates can lead to an output which is more than just "slightly wrong:" either one guard can see all of $T$ or not.

In order to circumvent this effect, CGAL supports the use of a variety of exact number types instead of floating-point arithmetic. Concretely, we resort to CGAL's `CGAL::Exact_predicates_exact_constructions_kernel` kernel [121]. It initially represents a coordinate by two doubles that encode an interval containing the actual coordinate. This suffices for many decisions, for instance, a comparison with another coordinate. However, in cases in which this is insufficient, e.g., because two intervals overlap, the exact coordinates are computed with the GNU Multiple Precision Arithmetic Library (GMP) [68]. Compared to the pure exact approach this usually yields a significant advantage regarding speed and memory [121].

## 24.3 Experiments

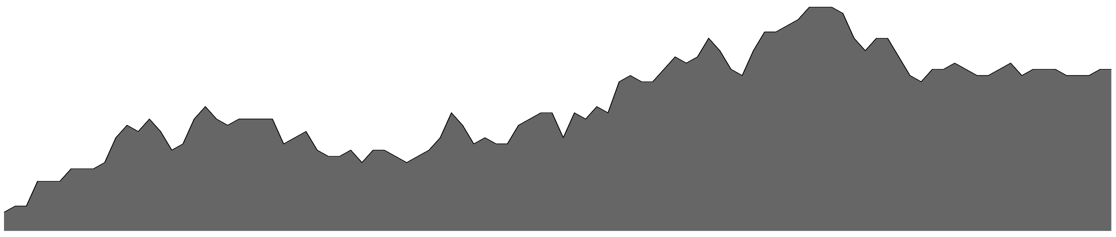
Algorithm 24.1 can solve large instances within minutes on a standard desktop computer; our filtering techniques prove critical to success. Especially EdgeFilter and WitnessFilter, see Sections 22.4.2 and 22.4.3, significantly increase the solvable instance size. Our instances, the tested configurations of Algorithm 24.1, the experimental setup, and our findings are described in Sections 24.3.1–24.3.4.

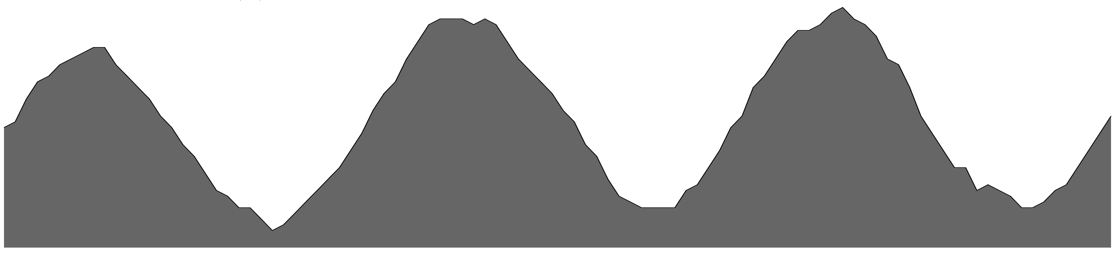
### 24.3.1 Instances

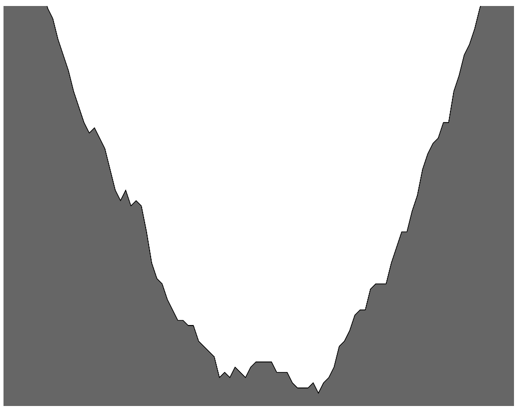
We test four classes of random terrains, see Figure 24.2 for an overview. Each class comprises 20 instances with $10^3$, $10^4$, $10^5$, $5 \cdot 10^5$, and $10^6$ vertices each, yielding a total of 400 instances. With reproducibility and possible future work in mind, we made these instances and additional ones, verified optimal solutions w.r.t. the VTGP and the CTGP where available, and the scripts generating the instances publicly available in the Terrain Guarding Problem Instance Library (TGPIL) [59].

A Walk, see Figure 24.2(a), has $n$ vertices with $x$-coordinates $0, \ldots, n-1$ and the $(i+1)$-th $y$-coordinate is a random offset from the $i$-th. SineWalk and ParabolaWalk, see Figures 24.2(b) and 24.2(c), are the sum of a Walk and a properly scaled sine or parabola, respectively. Both classes pose a challenge because many points located on a slope of a valley see a large part of the opposite slope that is highly fragmented by shadows of local features.

Preliminary experiments revealed that the above classes hardly require non-vertex guards for optimal solutions w.r.t. the CTGP. Hence we propose the ConcaveValleys class, see Figure 24.2(d). It encourages point guards because instances are constructed as follows: Start with a Walk instance. Iteratively pick an edge uniformly at random and replace it by a valley with concave slopes. Connect the slopes by a bottom edge, such that a point in its interior covers both slopes like in Figure 21.1(b). An optimal solution


**(a)** Walk: Random walk with uniform step width.


**(b)** SineWalk: Sum of a sine wave and a random walk.


**(c)** ParabolaWalk: Sum of a parabola and a random walk.

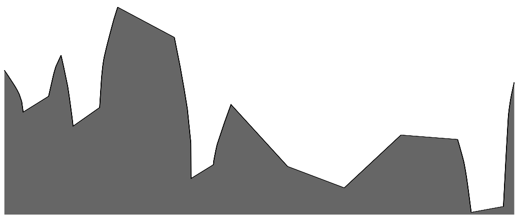

**(d)** ConcaveValleys: Optimal solutions require point guards in the valleys.

**Figure 24.2:** Four classes of random test instances available in the TGPIL [59].

| Configuration | Mode | EdgeFilter | DominationFilter | WitnessFilter |
|---|---|---|---|---|
| VDefault | VertexGuards | n/a | yes | yes |
| VNoDom | VertexGuards | n/a | no | yes |
| VNoW | VertexGuards | n/a | yes | no |
| PDefault | PointGuards | yes | yes | yes |
| PNoEdge | PointGuards | no | yes | yes |
| PNoDom | PointGuards | yes | no | yes |
| PNoW | PointGuards | yes | yes | no |

**Table 24.1:** Algorithm configurations, VTGP above and CTGP below.

w.r.t. the CTGP for such a terrain usually requires at least one guard in a bottom edge's interior.

None of the above classes deliberately provokes the NP-hardness of the TGP; they are not designed to contain a reduction of hard instances of, e.g., PLANAR 3SAT, as used in the NP-hardness proof of King and Krohn [87]. As motivated above, testing such instances is out of scope: We provide and evaluate the means to transform a terrain into a small discretization that can be handed to a solver; IP, PTAS, SAT, or other. The transformation has to be efficient and our experiments are designed to test just that. Combinatorially hard instances merely benchmark the underlying solver.

### 24.3.2 Configurations

We test Algorithm 24.1 in seven configurations, see Table 24.1, to individually assess the impact of each filtering technique from Section 22.4. VDefault, VNoDom, and VNoW test the VertexGuards mode; PDefault, PNoEdge, PNoDom, and PNoW test the considerably harder PointGuards mode. Recall that EdgeFilter does not apply to VertexGuards mode. Since we test 400 instances, this results in 2800 test runs.

### 24.3.3 Experimental Setup

We used eight identical Linux 3.13 machines with Intel Core i7-3770 CPUs running at 3.4 GHz, provided with 8 MB of cache and 16 GB of main memory. Every run was limited to 15 minutes of CPU time and 14 GB of memory. Our software, except solving IPs with CPLEX, is not parallelized. Refer to Section 24.2.1 for details regarding the toolchain and the implementation.

### 24.3.4 Results

The solution rate and median CPU time of every combination of configuration, instance class, and instance complexity are listed in Tables 24.2 and 24.3; each cell corresponds to 20 test runs. Due to the imposed time and memory limits, not all test runs succeeded; we account for unfinished test runs with an infinite completion time. Hence, we use the median instead of the mean throughout the analysis. More detailed timing information is presented in Figure 24.5.

| Configuration | Instance | #vertices $10^3$ | $10^4$ | $10^5$ | $5 \cdot 10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| VDefault | Walk | 100 % | 100 % | 100 % | 100 % | 100 % |
| | SineWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 100 % | 100 % |
| VNoDom | Walk | 100 % | 100 % | 100 % | 100 % | 100 % |
| | SineWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 100 % | 100 % |
| VNoW | Walk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | SineWalk | 100 % | 100 % | 0 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 25 % | 0 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 0 % | 0 % |
| PDefault | Walk | 100 % | 100 % | 100 % | 100 % | 100 % |
| | SineWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 100 % | 100 % |
| PNoEdge | Walk | 100 % | 100 % | 100 % | 55 % | 0 % |
| | SineWalk | 100 % | 100 % | 0 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 100 % | 0 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 90 % | 0 % |
| PNoDom | Walk | 100 % | 100 % | 100 % | 100 % | 100 % |
| | SineWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 100 % | 100 % |
| PNoW | Walk | 100 % | 100 % | 100 % | 0 % | 0 % |
| | SineWalk | 100 % | 100 % | 0 % | 0 % | 0 % |
| | ParabolaWalk | 100 % | 20 % | 0 % | 0 % | 0 % |
| | ConcaveValleys | 100 % | 100 % | 100 % | 0 % | 0 % |

**Table 24.2:** Solution rate by configuration, instance class, and instance complexity. Each cell corresponds to 20 test runs.

Except in PNoEdge mode, all unsolved instances were caused by running out of memory, we discuss this phenomenon below. In the following, we turn our attention to the relative hardness of the instance classes, an overview of VertexGuards and PointGuards modes, the impact of EdgeFilter, DominationFilter and WitnessFilter, timing behavior, and memory consumption.

**Overview: Instances** Tables 24.2 and 24.3 clearly reveal that the SineWalk and ParabolaWalk instances are harder to solve than their Walk and ConcaveValleys counterparts. This is to be expected, since SineWalk and ParabolaWalk contain a Walk as additive noise and since the sole purpose of ConcaveValleys is to encourage placing non-vertex guards, see Section 24.3.1. Furthermore, SineWalk and Parabola-Walk comprise facing valleys, resulting in highly fragmented visibility regions and complex visibility overlays in which a large portion of the guards and witnesses cannot be filtered out. This induces time and memory intensive calculations and a complex IP,

| Configuration | Instance | #vertices | | | | |
|---|---|---|---|---|---|---|
| | | $10^3$ | $10^4$ | $10^5$ | $5 \cdot 10^5$ | $10^6$ |
| VDefault | Walk | 0.0 s | 0.2 s | 2.9 s | 18.0 s | 40.3 s |
| | SineWalk | 0.0 s | 0.9 s | 13.8 s | n/a | n/a |
| | ParabolaWalk | 0.1 s | 1.8 s | 21.7 s | n/a | n/a |
| | ConcaveValleys | 0.2 s | 2.4 s | 24.2 s | 177.2 s | 337.6 s |
| VNoDom | Walk | 0.0 s | 0.3 s | 3.6 s | 21.8 s | 48.6 s |
| | SineWalk | 0.1 s | 1.3 s | 18.2 s | n/a | n/a |
| | ParabolaWalk | 0.1 s | 2.5 s | 27.8 s | n/a | n/a |
| | ConcaveValleys | 0.2 s | 2.4 s | 24.3 s | 177.1 s | 411.0 s |
| VNoW | Walk | 0.0 s | 0.4 s | 8.8 s | n/a | n/a |
| | SineWalk | 0.1 s | 6.4 s | n/a | n/a | n/a |
| | ParabolaWalk | 0.4 s | n/a | n/a | n/a | n/a |
| | ConcaveValleys | 0.2 s | 2.7 s | 32.0 s | n/a | n/a |
| PDefault | Walk | 0.0 s | 0.3 s | 4.6 s | 27.9 s | 62.4 s |
| | SineWalk | 0.1 s | 1.7 s | 26.3 s | n/a | n/a |
| | ParabolaWalk | 0.2 s | 3.3 s | 45.1 s | n/a | n/a |
| | ConcaveValleys | 0.1 s | 0.8 s | 10.3 s | 68.2 s | 137.4 s |
| PNoEdge | Walk | 0.2 s | 4.5 s | 79.7 s | 833.3 s | n/a |
| | SineWalk | 1.0 s | 58.8 s | n/a | n/a | n/a |
| | ParabolaWalk | 4.1 s | 220.3 s | n/a | n/a | n/a |
| | ConcaveValleys | 0.2 s | 3.2 s | 58.3 s | 652.3 s | n/a |
| PNoDom | Walk | 0.0 s | 0.4 s | 5.0 s | 31.5 s | 70.4 s |
| | SineWalk | 0.1 s | 2.0 s | 31.1 s | n/a | n/a |
| | ParabolaWalk | 0.2 s | 4.0 s | 51.8 s | n/a | n/a |
| | ConcaveValleys | 0.1 s | 0.9 s | 10.7 s | 68.0 s | 145.5 s |
| PNoW | Walk | 0.0 s | 0.6 s | 10.4 s | n/a | n/a |
| | SineWalk | 0.1 s | 7.8 s | n/a | n/a | n/a |
| | ParabolaWalk | 0.6 s | n/a | n/a | n/a | n/a |
| | ConcaveValleys | 0.1 s | 1.1 s | 20.8 s | n/a | n/a |

**Table 24.3:** Median CPU time by configuration, instance class, and instance complexity. Each cell corresponds to 20 test runs.

| Configuration | Instance | #vertices | | | | |
|---|---|---|---|---|---|---|
| | | $10^3$ | $10^4$ | $10^5$ | $5 \cdot 10^5$ | $10^6$ |
| PNoDom | Walk | 80.1 % | 86.8 % | 89.5 % | 91.0 % | 91.7 % |
| | SineWalk | 92.7 % | 97.6 % | 98.3 % | n/a | n/a |
| | ParabolaWalk | 97.5 % | 98.8 % | 98.9 % | n/a | n/a |
| | ConcaveValleys | 65.8 % | 72.5 % | 77.7 % | 79.9 % | 80.6 % |

**Table 24.4:** The median percentage of guard candidates removed by EdgeFilter.

making SineWalk and ParabolaWalk challenging instance classes.

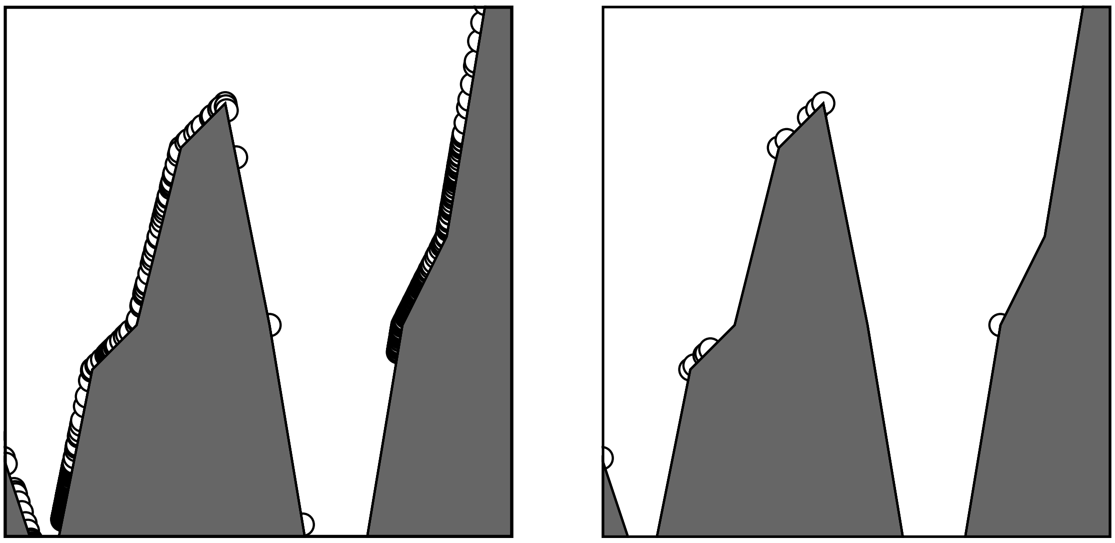
**Overview: Vertex Guards** VDefault and VNoDom solve all instances of Walk and ConcaveValleys, and the SineWalk and ParabolaWalk instances of up to $10^5$ vertices; VNoW can solve instances which are smaller by about a factor of 10, see Table 24.2. This already demonstrates the importance of WitnessFilter. VDefault and VNoDom require a comparable amount of CPU time, with a slight advantage for VDefault, see Table 24.3; VNoW is slower.

**Overview: Point Guards** In terms of solved instances, refer to Table 24.2, PDefault and PNoDom are the strongest configurations, solving all Walk and ConcaveValleys instances as well as the SineWalk and ParabolaWalk instances with up to $10^5$ vertices. PNoW and PNoEdge are much weaker. Table 24.3 indicates that PDefault is slightly faster than PNoDom. It is clear from the performance of PNoEdge that EdgeFilter is crucial in PointGuards mode.

**Impact of Filtering Edge-Interior Guards** Recall that EdgeFilter, see Section 22.4.2, only applies to PointGuards mode. Table 24.4 depicts the percentage of guards it removes in the PNoDom configuration — the only configuration without interfering guard filters — and Figure 24.3 illustrates its effectiveness using a $10^5$-vertex ParabolaWalk as example.

EdgeFilter proves to be our most effective guard filter by removing roughly 90 % (80 %) of the guard candidates in the $10^6$ vertex Walk (ConcaveValleys) instances, and well above 95 % in the largest solved SineWalk and ParabolaWalk instances. Tables 24.2 and 24.3 demonstrate that EdgeFilter massively improves performance in terms of solution rates and median CPU time. This makes it the key success factor when solving the CTGP, removing the computational barrier between the VTGP and the CTGP: Without it, PNoEdge would be the state of the art, and solvable instances of the CTGP would be smaller by at least a factor of 10 than for the VTGP and would take much more time.

**Impact of Filtering Dominated Guards** Table 24.5 displays the percentage of guard candidates that DominationFilter, refer to Section 22.4.1, filtered out in the VDefault and PNoEdge configurations. Observe that we need to obtain these numbers in configurations without other active guard filters.



**(a)** Unfiltered guard candidates. **(b)** Filtered guard candidates.

**Figure 24.3:** The effect of EDGEFILTER (excerpt of the right slope of a $10^5$-vertex PARABOLAWALK). Circles depict guard candidates. Local features on the right slope (e.g., the left peak) can induce many guard candidates (e.g., on the right): A feature casts a slightly different shadow for each vertex on the left slope (not depicted), resulting in many guard candidates. EDGEFILTER mitigates this effect very well.

| Configuration | Instance | #vertices | | | | |
|---|---|---|---|---|---|---|
| | | $10^3$ | $10^4$ | $10^5$ | $5 \cdot 10^5$ | $10^6$ |
| VDEFAULT | WALK | 65.8 % | 64.1 % | 63.6 % | 63.5 % | 63.5 % |
| | SINEWALK | 58.5 % | 63.0 % | 63.6 % | n/a | n/a |
| | PARABOLAWALK | 59.9 % | 63.3 % | 63.6 % | n/a | n/a |
| | CONCAVEVALLEYS | 13.4 % | 13.1 % | 13.3 % | 13.3 % | 13.3 % |
| PNOEDGE | WALK | 92.9 % | 94.7 % | 95.6 % | 95.8 % | n/a |
| | SINEWALK | 88.1 % | 96.7 % | n/a | n/a | n/a |
| | PARABOLAWALK | 93.2 % | 98.0 % | n/a | n/a | n/a |
| | CONCAVEVALLEYS | 77.1 % | 80.5 % | 83.9 % | 85.0 % | n/a |

**Table 24.5:** Median percentage of guard candidates removed by DOMINATIONFILTER.

| Configuration | Instance | #vertices $10^3$ | $10^4$ | $10^5$ | $5 \cdot 10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| VDefault | Walk | 91.9 % | 94.3 % | 95.4 % | 96.1 % | 96.4 % |
| | SineWalk | 95.8 % | 98.8 % | 99.3 % | n/a | n/a |
| | ParabolaWalk | 98.6 % | 99.4 % | 99.5 % | n/a | n/a |
| | ConcaveValleys | 76.5 % | 80.5 % | 83.7 % | 85.1 % | 85.5 % |
| PDefault | Walk | 91.9 % | 94.3 % | 95.5 % | 96.1 % | 96.4 % |
| | SineWalk | 96.3 % | 98.9 % | 99.3 % | n/a | n/a |
| | ParabolaWalk | 98.7 % | 99.5 % | 99.5 % | n/a | n/a |
| | ConcaveValleys | 80.3 % | 84.2 % | 87.4 % | 88.7 % | 89.1 % |

**Table 24.6:** Median percentage of witnesses removed by WitnessFilter.

DominationFilter has no impact on the solution rates within the limits imposed by our setup (see Section 24.3.3). VDefault and PDefault are slightly faster than VNoDom and PNoDom, respectively. The key advantage of DominationFilter, however, is that it saves memory by deleting dominated guard candidates; this is important since memory consumption is the bottleneck of our implementation, see below.

**Impact of Filtering Witnesses** The percentage of witnesses removed by WitnessFilter, see Section 22.4.3, in the VDefault and PDefault configurations is displayed in Table 24.6. Throughout our instances, WitnessFilter removes the vast majority of witnesses, often more than 95 %. Furthermore, Table 24.2 clearly shows that disabling it reduces the solvable instance size by at least a factor of 10. Given that PointGuards mode may have to deal with $|W(U)| \in \Theta(n^3)$ witnesses, this is not surprising. All of this renders WitnessFilter simple, fast, and useful.

**Timing Behavior** Figures 24.4 and 24.5 show how much CPU time is spent in each part of Algorithm 24.1 and the distribution of the CPU time, respectively. For a comparison, we pick an instance class and a complexity that was solved by every configuration, a Walk with $10^5$ vertices, the other combinations are omitted as they permit the same interpretation.

The strongest impact is that of EdgeFilter, which is disabled in the rightmost bar in Figure 24.4. Without EdgeFilter there is a computational gap between PointGuards and VertexGuards mode, as determining visibility regions of unneeded guards dominates the CPU time. Guard-filtering time also increases in PNoEdge mode because DominationFilter is active and has more guards to compare.

WitnessFilter has the second most important impact. In its absence, CPU time roughly doubles due to an increased IP solution time: In the IP, non-inclusion-minimal witnesses form constraints that are dominated by the inclusion-minimal witnesses' constraints. The IP solver eliminates dominated constraints in a preprocessing phase, but WitnessFilter does so more efficiently, since it exploits the underlying geometry.

The timing behavior is hardly influenced by DominationFilter. It is, however, beneficial w.r.t. memory consumption, see below.

A general observation is that the filtering mechanisms significantly reduce the com-

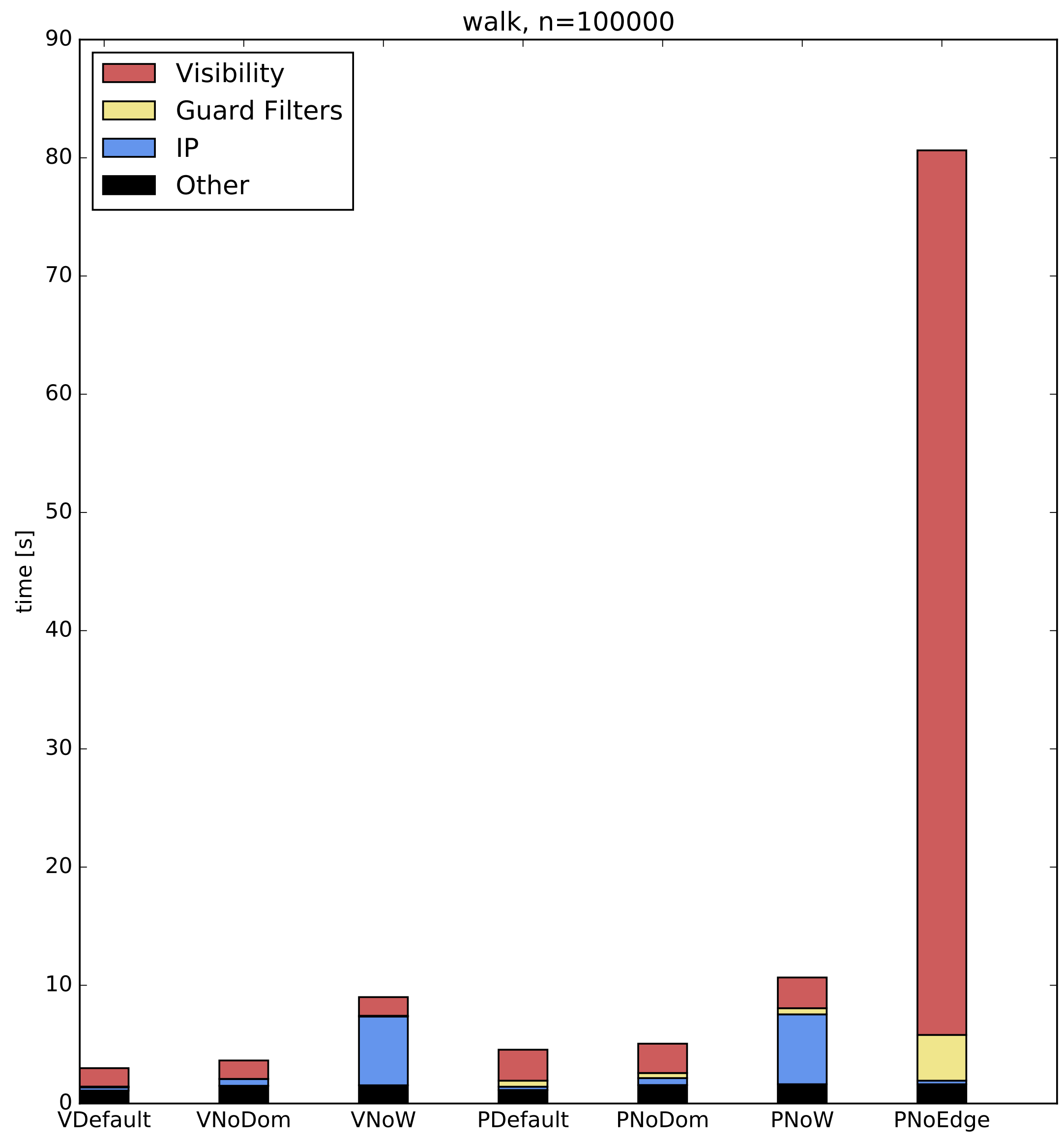


**Figure 24.4:** Median CPU time by subroutine regarding $10^5$-vertex WALK instances.

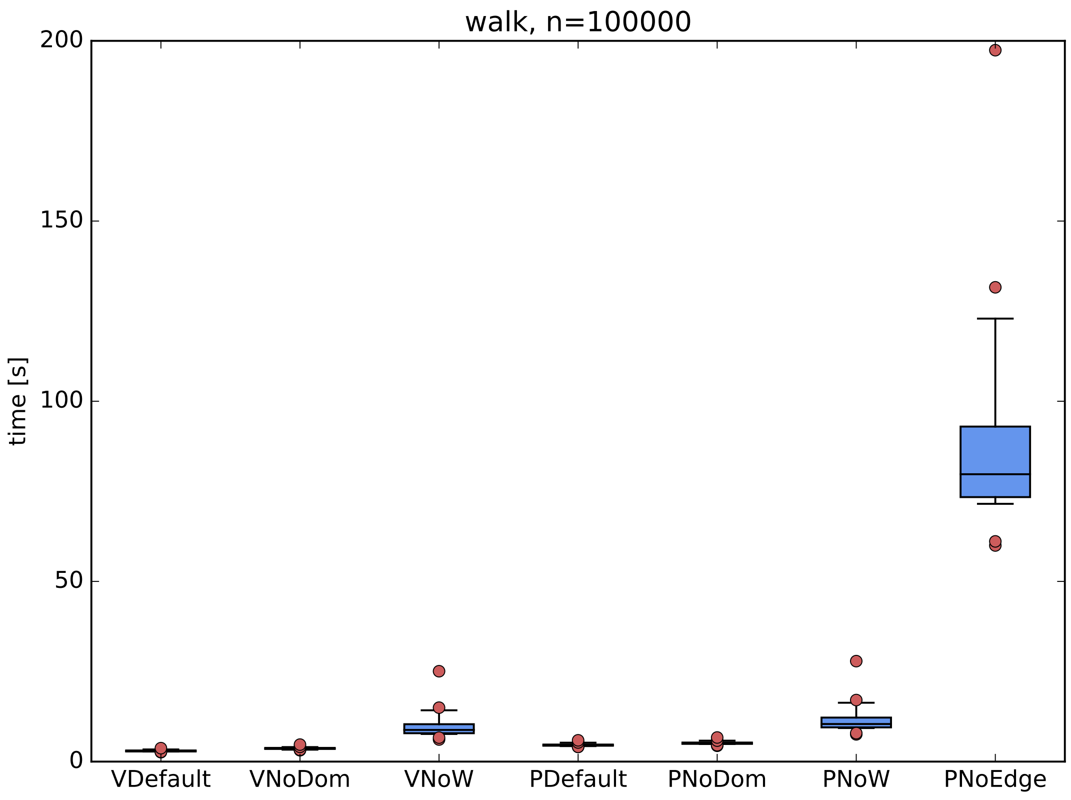


**Figure 24.5:** Box plot of CPU time regarding $10^5$-vertex Walk instances. Boxes depict 1st–3rd quartile, whiskers the 10th and 90th percentile.

putational overhead. One would expect the IP solution-time to dominate an exact solver for an NP-hard problem. This, however, tends not to be the case in geometric optimization problems like the AGP and its relatives [40], an effect that is rooted in the EGC paradigm: Instead of floating-point arithmetic, exact number types must be used to ensure consistent and correct results. VDefault and PDefault still are not clearly dominated by the IP solution time, but much closer to it than unfiltered approaches.

**Memory Consumption** Within our experimental setup, using the VDefault and PDefault modes, even instances with $10^6$ vertices are solved within minutes. All unsolved configuration–instance pairs run out of memory, not time. The only exception is PNoEdge, which occasionally runs out of time owed to the large number of unnecessary visibility calculations otherwise prevented by EdgeFilter, see Table 24.4 and the corresponding discussion. So as long as instances are not designed to reveal the NP-hardness of the TGP, the limiting resource is memory.

Two phases of Algorithm 24.1 generate a significant amount of data that persists in memory. One phase is the computation of all visibility regions of all vertices $V$ in line 4. This may store $\Theta(n^2)$ $x$-coordinates[3] in memory which define the unfiltered guard-candidate set $U$. Filters remove the vast majority of candidates from $U$. The other phase is the one that determines the visibility regions of the remaining points in $U \setminus V$, generating the largest chunk of data in line 12: At this point, we may keep $\Theta(n^3)$ $x$-coordinates in memory. It is not obvious how to avoid holding all these visibility regions in memory since a guard at the far right of the terrain may still see a region at its very left; it is an open question if it is possible to apply WitnessFilter before knowing all extremal points.

We remark that the memory bottleneck may be amplified by the fact that we follow the EGC paradigm, compare Section 24.2.1, which ensures a correct and consistent representation of all visibility regions and a correct order of all visibility events. Specifically, we do not store coordinates of points using floating-point arithmetic. In our case, the `CGAL::Exact_predicates_exact_constructions_kernel` kernel [121] performed better in terms of CPU time and memory usage than non-lazy variants like GMP [68].

[3]We never store $y$-coordinates of visibility regions. Instead, we represent them as unions of intervals of $x$-coordinates.

# CHAPTER 25

# Conclusion

We present a discretization with $O(n^2)$ guard candidates and $O(n^3)$ witnesses for the CTGP. It was discovered independently from that of King [86] and is asymptotically smaller by a factor of $n$. This positively answers King's question whether $O(n^2)$ guard candidates suffice. The existence of a polynomially sized discretization settles two open questions: (1) The decision variant of the CTGP is a member of $\mathsf{NP}$ and, since its $\mathsf{NP}$-hardness is known [87], $\mathsf{NP}$-complete. (2) Moreover, the CTGP admits a PTAS, since the PTAS for the discrete TGP [66] applies to our discretization.

Furthermore, we propose an algorithm for finding optimal solutions for the CTGP and the VTGP; it directly generalizes to other discrete versions of the TGP. Our implementation solves instances with up to $10^6$ vertices within minutes. The filtering techniques reducing the size of the discretization — and with it the geometric overhead — are a key success factor, essentially removing the computational barrier between the CTGP and the VTGP.

# List of Figures

*"Their art was the art that convinced — when we saw the pictures we saw the daemons themselves and were afraid of them."*

H. P. Lovecraft, Pickman's Model

# List of Tables

*“Ich seh’ Gauß. . . ”*

Matthias Függer

# List of Algorithms

*"Beware of bugs in the above code; I have only proved it correct, not tried it."*

Donald Knuth

# List of Acronyms

*"Abbrv. is the abbrv. for abbrv."*

unknown

| | |
|---|---|
| **3SAT** | Boolean 3-Satisfiability Problem [63] |
| **ADC** | Analog-to-Digital Converter |
| **AGP** | Art Gallery Problem [117] |
| **APSP** | All-Pairs Shortest Paths (Example 14.6) |
| **APWP** | All-Pairs Widest Paths (Example 14.15) |
| **ASIC** | Application-Specific Integrated Circuit |
| **BFS** | Breadth-First Search |
| **BRGC** | Binary Reflected Gray Code (Section 10.2) |
| **CGAL** | Computational Geometry Algorithms Library [28] |
| **CMOS** | Complementary Metal-Oxide-Semiconductor [4, 131] |
| **CMUX** | Metastability-Containing Multiplexer (Equation (6.5)) |
| **CRCW** | Concurrent Read Concurrent Write [77] |
| **CREW** | Concurrent Read Exclusive Write [77] |
| **CTGP** | Continuous Terrain Guarding Problem (Definition 21.1) |
| **DAG** | Directed Acyclic Graph |
| **DCO** | Digitally Controlled Oscillator |
| **DNF** | Disjunctive Normal Form |
| **EGC** | Exact Geometric Computation [121] |
| **EREW** | Exclusive Read Exclusive Write [77] |
| **FET** | Field-Effect Transistor [4, 131] |
| **FPGA** | Field-Programmable Gate Array [131] |
| **FRT** | Fakcharoenphol, Rao, and Talwar [50] (Definition 17.2) |
| **FPT** | Fixed-Parameter Tractable |
| **GMP** | GNU Multiple Precision Arithmetic Library [68] |
| **ID** | Identifier |
| **IP** | Integer Linear Program |
| **$k$-DSDP** | $k$-Distinct-Shortest Distance Problem (Definition 14.22) |
| **$k$-SDP** | $k$-Shortest Distance Problem (Definition 14.22) |

| | |
|---|---|
| ***k*-SSP** | $k$-Source Shortest Paths (Example 14.5) |
| **LE** | Least Element (Equation (17.11)) |
| **LP** | Linear Program |
| **MBF** | Moore-Bellman-Ford [16, 54, 116] (Algorithm 13.1) |
| **MBF-like** | Moore-Bellman-Ford-like (Definition 13.11) |
| **MSSP** | Multi-Source Shortest Paths (Example 14.7) |
| **MST** | Minimum Spanning Tree |
| **MSWP** | Multi-Source Widest Paths (Example 14.16) |
| **MTBF** | Mean Time Between Failures |
| **MUX** | Multiplexer (Equation (6.2)) |
| **PRAM** | Parallel Random-Access Machine [77] |
| **PTAS** | Polynomial-Time Approximation Scheme |
| **SAT** | Boolean Satisfiability Problem [63] |
| **SC** | Set Cover [63] |
| **SLF** | Simple Linear Function (Definition 13.13) |
| **SPD** | Shortest-Path Diameter (Equation (12.10)) |
| **SPICE** | Simulation Program with Integrated Circuit Emphasis [127] |
| **SSSP** | Single-Source Shortest Paths (Example 14.4) |
| **SSWP** | Single-Source Widest Paths (Example 14.14) |
| **TC** | Thermometer Code (Section 10.2) |
| **TDC** | Time-to-Digital Converter |
| **TGP** | Terrain Guarding Problem (Definition 21.1) |
| **TGPIL** | Terrain Guarding Problem Instance Library [59] |
| **TSP** | Traveling Salesman Problem [63] |
| **TTP** | Time-Triggered Protocol |
| **VLSI** | Very Large-Scale Integration |
| **VTGP** | Terrain Guarding Problem with Vertex Guards (Definition 21.1) |
| **WPP** | Widest Path Problem (Section 14.2) |

# Bibliography

*"Quotations — the karaoke of ideas."*

John Oliver, Last Week Tonight

[1] M. Abrahamsen, A. Adamaszek, and T. Miltzow. Irrational guards are sometimes needed. *CoRR*, abs/1701.05475, 2017.

[2] P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. North-Holland, Amsterdam, first edition, 2000.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1983.

[4] P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design.* The Oxford Series in Electrical and Computer Engineering. Oxford University Press USA, third edition, 2011.

[5] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Science*, 54(2):255–262, 1997.

[6] N. Alon, R. M. Karp, D. Peleg, and D. B. West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

[7] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 542–547, 1997.

[8] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[9] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 161–168, 1998.

[10] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[11] K. E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies (AFIPS)*, pages 307–314, 1968.

[12] R. Becker, A. Karrenbauer, S. Krinninger, and C. Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *CoRR*, abs/1607.05127, 2016.

[13] S. Beer and R. Ginosar. Eleven ways to boost your synchronizer. *IEEE Transactions on VLSI Systems*, 23(6):1040–1049, 2015.

[14] S. Beer, R. Ginosar, J. Cox, T. Chaney, and D. M. Zar. Metastability challenges for 65nm and beyond: simulation and measurements. In *Design, Automation and Test in Europe (DATE)*, pages 1297–1302, 2013.

[15] S. Beer, R. Ginosar, M. Priel, R. R. Dobkin, and A. Kolodny. The devolution of synchronizers. In *16th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 94–103, 2010.

[16] R. E. Bellman. On a routing problem. *Quarterly Applied Mathematics*, 16:87–90, 1958.

[17] R. Belschner, J. Berwanger, F. Bogenberger, C. Ebner, H. Eisele, B. Elend, T. M. Forest, T. Führer, P. Fuhrmann, F. Hartwich, et al. FlexRay communication protocol, 2003. Patent Application EP2002/0008171.

[18] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5d terrain guarding. *SIAM Journal on Computing*, 36(6):1631–1647, 2007.

[19] G. E. Blelloch, Y. Gu, J. Shun, and Y. Sun. Parallelism in randomized incremental algorithms. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 467–478, 2016.

[20] G. E. Blelloch, Y. Gu, and Y. Sun. Efficient construction of probabilistic tree embeddings. *CoRR*, abs/1605.04651, 2016.

[21] G. E. Blelloch, A. Gupta, and K. Tangwongsan. Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 205–213, 2012.

[22] G. E. Blelloch and K. Tangwongsan. Parallel approximation algorithms for facility-location problems. In *Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 315–324, 2010.

[23] boost C++ libraries. `http://www.boost.org/`.

[24] J. A. Brzozowski, Z. Ésik, and Y. Iland. Algebras for hazard detection. In *31st IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, pages 3–12, 2001.

[25] J. A. Brzozowski and M. Yoeli. On a ternary model of gate networks. *IEEE Transactions on Computers*, C-28(3):178–184, March 1979.

[26] J. Bund, C. Lenzen, and M. Medina. Near-optimal metastability-containing sorting networks. In *Design, Automation and Test in Europe (DATE)*, pages 226–231, 2017.

[27] F. Bungiu, M. Hemmer, J. Hershberger, K. Huang, and A. Kröller. Efficient computation of visibility polygons. *CoRR*, abs/1403.3905, 2014.

[28] The computational geometry algorithms library (CGAL). `http://www.cgal.org/`.

[29] A. K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24(1):7–11, 1978.

[30] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[31] K. Chwa, B. Jo, C. Knauer, E. Moet, R. van Oostrum, and C. Shin. Guarding art galleries by guarding witnesses. *International Journal of Computational Geometry & Applications*, 16(2-3):205–226, 2006.

[32] K. L. Clarkson and K. R. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[33] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997.

[34] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of the ACM*, 47(1):132–166, 2000.

[35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, third edition, 2009.

[36] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *International Transactions in Operational Research*, 18(4):425–448, 2011.

[37] IBM ILOG CPLEX Optimization Studio. `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`.

[38] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

[39] V. De Heyn, G. Van der Plas, J. Ryckaert, and J. Craninckx. A fast start-up 3GHz–10GHz digitally controlled oscillator for UWB impulse radio in 90nm CMOS. In *European Solid State Circuits Conference*, pages 484–487, September 2007.

[40] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering art galleries. In *Algorithm Engineering – Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 379–417. Springer, 2016.

[41] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[42] S. Durocher, P. C. Li, and S. Mehrabi. Guarding orthogonal terrains. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015.

[43] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, March 1965.

[44] S. Eidenbenz. Approximation algorithms for terrain guarding. *Information Processing Letters*, 82(2):99–105, 2002.

[45] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[46] K. M. Elbassioni, E. Krohn, D. Matijević, J. Mestre, and D. Ševerdija. Improved approximations for guarding 1.5-dimensional terrains. *Algorithmica*, 60(2):451–463, 2011.

[47] M. Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal on Computing*, 36(2):433–456, 2006.

[48] M. Elkin and O. Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 128–137, 2016.

[49] M. Ernestus, S. Friedrichs, M. Hemmer, J. Kokemüller, A. Kröller, M. Moeini, and C. Schmidt. Algorithms for art gallery illumination. *Journal Global Optimization*, 68(1):23–45, 2017.

[50] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[51] S. P. Fekete, S. Friedrichs, A. Kröller, and C. Schmidt. Facets for art gallery problems. *Algorithmica*, 73(2):411–440, 2015.

[52] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[53] S. Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.

[54] L. R. Ford. Network flow theory. Technical report, The RAND Corporation, 1956.

[55] S. Friedrichs, M. Függer, and C. Lenzen. Metastability-containing circuits. *CoRR*, abs/1606.06570, 2016.

[56] S. Friedrichs, M. Hemmer, J. King, and C. Schmidt. The continuous 1.5d terrain guarding problem: Discretization, optimal solutions, and PTAS. *Journal of Computational Geometry*, 7(1):256–284, 2016.

[57] S. Friedrichs, M. Hemmer, and C. Schmidt. A PTAS for the continuous 1.5d terrain guarding problem. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG)*, pages 367–373, 2014.

[58] S. Friedrichs, M. Hemmer, and C. Schmidt. Exact solutions for the continuous terrain guarding problem. In *31st European Workshop on Computational Geometry (EuroCG)*, pages 212–215, 2015.

[59] S. Friedrichs, M. Hemmer, and C. Schmidt. Terrain guarding problem instance library. `http://resources.mpi-inf.mpg.de/tgp/index.html#instances`, August 2015. Version 2015-08-06.

[60] S. Friedrichs and A. Kinali. Efficient metastability-containing multiplexers. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017. To appear.

[61] S. Friedrichs and C. Lenzen. Parallel metric tree embedding based on an algebraic view on Moore-Bellman-Ford. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 455–466, 2016.

[62] M. Függer, A. Kinali, C. Lenzen, and T. Polzer. Metastability-aware memory-efficient time-to-digital converters. In *23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2017.

[63] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[64] The GNU compiler collection. `http://gcc.gnu.org/`.

[65] M. Ghaffari and C. Lenzen. Near-optimal distributed tree embedding. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 197–211, 2014.

[66] M. Gibson, G. Kanade, E. Krohn, and K. R. Varadarajan. Guarding terrains via local search. *Journal of Computational Geometry*, 5(1):168–178, 2014.

[67] R. Ginosar. Fourteen ways to fool your synchronizer. In *9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 89–97, 2003.

[68] The GNU multiple precision arthmetic library. `https://gmplib.org/`.

[69] C. T. Gray, W. Liu, W. A. M. Van Noije, T. A. Hughes, and R. K. Cavin. A sampling technique and its CMOS implementation with 1Gb/s bandwidth and 25ps resolution. *IEEE Journal of Solid-State Circuits*, 29(3):340–349, 1994.

[70] A. Haas and M. Hemmer. Efficient algorithms and implementations for visibility in 1.5d terrains. In *31st European Workshop on Computational Geometry (EuroCG)*, pages 216–219, 2015.

[71] M. Hauptmann and M. Karpinski. A compendium on Steiner tree problems. `http://theory.cs.uni-bonn.de/info5/steinerkompendium/netcompendium.html`, 2015. Visited 2017-04-25.

[72] U. Hebisch and H. J. Weinert. *Semirings: Algebraic Theory and Applications in Computer Science.* Series in algebra. World Scientific, 1998.

[73] M. Henzinger, S. Krinninger, and D. Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 489–498, 2016.

[74] D. A. Huffman. The design and use of hazard-free switching networks. *Journal of the ACM*, 4(1):47–62, January 1957.

[75] F. Hurtado, M. Löffler, I. Matos, V. Sacristán, M. Saumell, R. I. Silveira, and F. Staals. Terrain visibility with multiple viewpoints. *International Journal of Computational Geometry & Applications*, 24(4):275–306, 2014.

[76] International technology roadmap for semiconductors. `http://www.itrs2.net/`, 2013.

[77] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[78] D. S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 67–161. Elsevier, 1990.

[79] J. Kahn, M. Klawe, and D. Kleitman. Traditional art galleries require fewer watchmen. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):194–206, 1983.

[80] M. J. Katz and G. S. Roisman. On guarding the vertices of rectilinear domains. *International Journal of Computational Geometry & Applications*, 39(3):219–228, 2008.

[81] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012.

[82] F. Khodakarami, F. Didehvar, and A. Mohades. A fixed-parameter algorithm for guarding 1.5d terrains. *Theoretical Computer Science*, 595:130–142, 2015.

[83] A. Kinali, F. Huemer, and C. Lenzen. Fault-tolerant clock synchronization with high precision. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 490–495, 2016.

[84] J. King. Errata on “A 4-approximation for guarding 1.5-dimensional terrains”. `http://www.cs.mcgill.ca/~jking/papers/4apx_latin.pdf`. Visited 2017-06-05.

[85] J. King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *7th Latin American Theoretical Informatics Symposium (LATIN)*, pages 629–640, 2006.

[86] J. King. *Guarding Problems and Geometric Split Trees.* PhD thesis, McGill University, 2010.

[87] J. King and E. Krohn. Terrain guarding is NP-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.

[88] D. J. Kinniment. *Synchronization and Arbitration in Digital Systems.* Wiley, 2008.

[89] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev. Synchronization circuit performance. *IEEE Journal of Solid-State Circuits*, 37(2):202–209, February 2002.

[90] S. C. Kleene. *Introduction to Metamathematics.* Van Nostrand, 1952.

[91] P. N. Klein and S. Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205–220, 1997.

[92] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[93] H. Kopetz, A. Krüger, D. Millinger, and A. V. Schedl. A synchronization strategy for a time-triggered multicluster real-time system. In *14th Symposium on Reliable Distributed Systems (SRDS)*, pages 154–161, 1995.

[94] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, 36(8):933–940, 1987.

[95] A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt. Exact solutions and bounds for general art gallery problems. *ACM Journal of Experimental Algorithmics*, 17(2):2.3:1–2.3:23, 2012.

[96] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[97] A. Laurentini. Guarding the walls of an art gallery. *The Visual Computer*, 15(6):265–278, 1999.

[98] D. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[99] C. Lenzen, M. Függer, A. Kinali, S. Friedrichs, and M. Medina. Efficient and dependable clock synchronization in hardware, 2016. Patent Application PCT/EP2016/002179.

[100] C. Lenzen and M. Medina. Efficient metastability-containing Gray code 2-sort. In *22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 49–56, 2016.

[101] C. Lenzen and B. Patt-Shamir. Fast routing table construction using small messages: extended abstract. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 381–390, 2013.

[102] C. Lenzen and B. Patt-Shamir. Improved distributed Steiner forest construction. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 262–271, 2014.

[103] C. Lenzen and B. Patt-Shamir. Fast partial distance estimation and applications. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2015.

[104] C. Lenzen and D. Peleg. Efficient distributed source detection with limited bandwidth. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013.

[105] A. Lumsdaine, D. P. Gregor, B. Hendrickson, and J. W. Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(1):5–20, 2007.

[106] J. Lundelius Welch and N. A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

[107] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[108] L. R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, 30(2):107–115, February 1981.

[109] G. Martinović, D. Matijević, and D. Ševerdija. Efficient parallel implementations of approximation algorithms for guarding 1.5d terrains. *Croatian Operational Research Review*, 6(1):79–89, 2015.

[110] M. Mendel and C. Schwob. Fast C-K-R partitions of sparse graphs. *Chicago Journal of Theoretical Computer Science*, 2009(2):1–18, 2009.

[111] M. Mendler, T. R. Shiple, and G. Berry. Constructive Boolean circuits and the exactness of timed ternary simulation. *Formal Methods in System Design*, 40(3):283–329, 2012.

[112] R. R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1-3):35–60, 2004.

[113] U. Meyer and P. Sanders. $\Delta$-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

[114] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[115] M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[116] E. F. Moore. The shortest path through a maze. In *Symposium on the Theory of Switching*, pages 87–90, 1959.

[117] J. O'Rourke. *Art Gallery Theorems and Algorithms.* International Series of Monographs on Computer Science. Oxford University Press, New York, 1987.

[118] J. O'Rourke and K. J. Supowit. Some NP-hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29(2):181–189, 1983.

[119] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, Philadelphia, 2000.

[120] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing*, 30(5):1427–1442, 2000.

[121] S. Pion and A. Fabri. A generic lazy evaluation scheme for exact geometric computations. *Science of Computer Programming*, 76(4):307–323, 2011.

[122] T. E. Rahkonen and J. T. Kostamovaara. The use of stabilized CMOS delay lines for the digitization of short time intervals. *IEEE Journal of Solid-State Circuits*, 28(8):887–894, 1993.

[123] G. W. Roberts and M. Ali-Bakhshian. A brief introduction to time-to-digital and digital-to-time converters. *IEEE Transactions on Circuits and Systems*, 57(3):153–157, 2010.

[124] D. Schuchardt and H. Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41:261–267, 1995.

[125] T. C. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, 1992.

[126] simple-svg. `http://code.google.com/p/simple-svg/`.

[127] Simulation program with integrated circuit emphasis (SPICE). `http://ngspice.sourceforge.net/`.

[128] G. Tarawneh, M. Függer, and C. Lenzen. Metastability tolerant computing. In *23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2017.

[129] G. Tarawneh and A. V. Yakovlev. An RTL method for hiding clock domain crossing latency. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 540–543, 2012.

[130] G. Tarawneh, A. V. Yakovlev, and T. S. T. Mak. Eliminating synchronization latency using sequenced latching. *IEEE Transactions on VLSI Systems*, 22(2):408–419, 2014.

[131] U. Tietze, C. Schenk, and E. Gamm. *Halbleiter-Schaltungstechnik.* Springer Vieweg, fifteenth edition, 2012.

[132] S. H. Unger. Hazards and delays in asynchronous sequential switching circuits. *IRE Transactions on Circuit Theory*, 6(1):12–25, Mar 1959.

[133] H. J. M. Veendrick. The behaviour of flip-flops used as synchronizers and prediction of their failure rate. *IEEE Journal of Solid-State Circuits*, 15(2):169–176, 1980.

[134] A. V. Yakovlev, M. Kishinevsky, A. Kondratyev, and L. Lavagno. OR causality: Modelling and hardware implementation. In *15th International Conference on Application and Theory of Petri Nets*, pages 568–587, 1994.

[135] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.