2017

Parallelism with limited nondeterminism

BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

PARALLELISM WITH LIMITED NONDETERMINISM

by

JEFFREY FINKELSTEIN

B.S., Tufts University, 2010

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
2017

Approved by

First Reader

Steven Homer, Ph.D.
Professor of Computer Science

Second Reader

Benjamin Hescott, Ph.D.
Assistant Professor of Computer Science
Tufts University

Third Reader

Péter Gács, Ph.D.
Professor of Computer Science

To my parents.

PARALLELISM WITH LIMITED NONDETERMINISM

JEFFREY FINKELSTEIN

Boston University Graduate School of Arts and Sciences, 2017

Major Professor: Steven Homer, Ph.D., Professor of Computer Science

ABSTRACT

Computational complexity theory studies which computational problems can be solved with limited access to resources. The past fifty years have seen a focus on the relationship between intractable problems and efficient algorithms. However, the relationship between inherently sequential problems and highly parallel algorithms has not been as well studied. Are there efficient but inherently sequential problems that admit some relaxed form of highly parallel algorithm? In this dissertation, we develop the theory of structural complexity around this relationship for three common types of computational problems.

Specifically, we show tradeoffs between time, nondeterminism, and parallelizability. By clearly defining the notions and complexity classes that capture our intuition for parallelizable and sequential problems, we create a comprehensive framework for rigorously proving parallelizability and non-parallelizability of computational problems. This framework provides the means to prove whether otherwise tractable problems can be effectively parallelized, a need highlighted by the current growth of multiprocessor systems. The views adopted by this dissertation—alternate approaches to solving sequential problems using approximation, limited nondeterminism, and parameterization—can be applied practically throughout computer science.

Contents

List of abbreviations

AC polylogarithmic depth, unbounded fan-in circuits

AP approximation-preserving

ApxNCO constant factor NC approximation

ApxPO constant factor polynomial-time approximation

BS-BD-CVP bounded size, bounded depth circuit value problem

CSP constraint satisfaction problem

DTIME deterministic time

DSPACE deterministic space

EXP deterministic exponential time

ENCAS efficient NC approximation scheme

FNC functions computable in NC

FNCAS fully NC approximation scheme

FO first-order formulas

FP functions computable in polynomial time

FPT fixed-parameter tractable

FPTAS fully polynomial-time approximation scheme

GC guess-and-check

HDS high degree subgraph

L logarithmic space

MaxNP nondeterministic polynomial-time maximization

MaxSNP strict nondeterministic polynomial-time maximization

NC polylogarithmic depth, bounded fan-in circuits

NCX constant factor approximable NC optimization

NCO NC optimization

NCAS NC approximation scheme

NEXP nondeterministic exponential time

NL nondeterministic logarithmic space

NNC nondeterministic NC

NNCO nondeterministic NC optimization

NP nondeterministic polynomial time

NPO nondeterministic polynomial time optimization

P polynomial time

p-k-CE parameterized circuit k-evaluation

p-BHP parameterized bounded halting problem

p-CSAT parameterized circuit satisfiability

p-k-NC^{d}CSAT parameterized NC^d circuit satisfiability

p-SCE parameterized small circuit evaluation

p-WCE parameterized weighted circuit evaluation

paraβL parameterized logarithmic space with $f(k)\log n$ one-way nondeterminism

paraAC parameterized AC

paraEP parameterized polynomial-time evaluation

paraFO parameterized first-order formulas

paraNC parameterized NC

paraWL parameterized logarithmic space with $f(k)\log n$ two-way nondeterminism

paraWNC parameterized NC with $f(k)\log n$ nondeterminism

paraWNL parameterized nondeterministic logarithmic space with $f(k)\log n$ two-way nondeterminism

paraWP parameterized polynomial time with $f(k)\log n$ two-way nondeterminism

pb polynomially bounded

PCP probabilistically checkable proofs

PNC parameterized analog of NC

PO polynomial-time optimization

poly polynomial functions

polyL polylogarithmic space

polylog polylogarithmic functions

PTAS polynomial-time approximation scheme

QP quasipolynomial time

RNC randomized NC

SL symmetric logarithmic space

SUBEXP subexponential time

W[NC^{d}SAT] closure of parameterized NC^d satisfiability

W[SAT] closure of parameterized formula satisfiability

W[P] closure of parameterized circuit satisfiability

Chapter 1

Introduction

Since parallel computing is again becoming a topic of interest in computer science, it is important to revisit the theoretical foundations of highly parallel computing. In theoretical computer science, computational complexity studies what problems can be solved when facing limited access to resources. With parallelism as the resource of interest, computational complexity has already classified numerous computational problems as either "inherently sequential" or "parallelizable." Inherently sequential computational problems, unlike parallizable problems, see no significant speedup when run on highly parallel computers.

Computational problems can be further classified into three types: decision problems, optimization problems, and parameterized problems. Decision problems are of the form "Does object x have property P?" Decision problems lead to the other two kinds of problems by modifying either the solution space or the resource usage bounds. Optimization problems generalize decision problems by allowing a search for a good solution among many candidates. Parameterized problems generalize decision problems by allowing resource bounds to depend on a parameter of the problem

instance instead of simply the size. The study of the computational complexity of both optimization problems and parameterized problems provides a more detailed view than the study of decision problems alone.

Just as there are efficient approximations for intractable optimization problems, so too are there efficient and highly parallel approximations for optimization problems that are tractable but inherently sequential. For example, the problem of computing the optimal vector in a positive linear program—a problem relevant to distributed flow control within a network of routers—is inherently sequential, but a vector very close to the optimal one can be computed quickly in parallel. Similarly, just as there are fixed-parameter tractable algorithms for some intractable problems, so too are there fixed-parameter parallel algorithms for some sequential parameterized problems. For example, the problem of evaluating a Boolean circuit on a given input is inherently sequential, but the circuit can be evaluated quickly in parallel when the depth of the circuit is considered a parameter of the problem. These facts, which require proofs, give practicioners the confidence that when faced with inherently sequential problems, all hope is not lost.

Our guiding question is whether there are efficient but inherently sequential problems that admit a "relaxed" highly parallel algorithm. For decision problems, are there inherently sequential problems that can be solved by highly parallel algorithms when augmented with a small amount of nondeterminism? For optimization problems, are there inherently sequential problems that can be approximated by highly parallel algorithms? For parameterized problems, are there inherently sequential problems that can be solved by highly parallel algorithms under certain parameterizations?

We develop the theory of structural complexity for highly parallel algorithms for tractable but inherently sequential problems in decision problems, optimization problems, and parameterized problems. This area has not been well-studied, and when

it has been studied, the results focus mostly on parallel algorithms for *intractable* problems, not parallel algorithms for tractable sequential problems . This dissertation proves the main theorems required for the study of the computational complexity of parallelizable versus sequential computational problems and provides the motivation and intuition to understand their significance and use.

There are three main chapters in this dissertation, each of which discusses a different type of computational problem, namely decision problems, optimization problems, and parameterized problems. Each chapter discusses the main approaches to proving the limitations of highly parallel algorithms for tractable but inherently sequential problems of the respective type. Further, we show how adding a limited amount of nondeterminism to a highly parallel algorithm allows us to circumvent some limitations of parallelism without requiring sequential computation.

In chapter 2, we discuss augmenting a highly parallel algorithm for decision problems with a small amount of nondeterminism as the basis for a technique to prove inapproximability of optimization problems. In chapter 3, we define and explore the complexity classes associated with parallel approximation algorithms for inherently sequential optimization problems. In chapter 4, we formalize the tradeoffs between time, parallelism, and nondeterminism in parameterized problems and show some connections with decision and optimization problems.

Our findings demonstrate that, under reasonable complexity theoretic assumptions, there are inherently sequential problems that do not admit parallel algorithms, and even parallel algorithms augmented with some additional resources or relaxed objectives. Furthermore, under the assumption that $\mathsf{NC} \neq \mathsf{P}$, the fact that $\mathsf{NNC}[\mathsf{poly}] = \mathsf{NP}$ ([68]) but $\mathsf{NNCO} \neq \mathsf{NPO}$ (Theorem 3.3.5) leads us to conclude that viewing a computational problem as merely a decision problem is too coarse-grained an approach—it does not give enough information about the computational complexity of the problem.

The conjecture that $\mathsf{paraWNC} \neq \mathsf{paraWP}$ if and only if $\mathsf{NNC}[\omega(\log n)] = \mathsf{NP}[\omega(\log n)]$ (Conjecture 4.5.2) supports this view as well, if the conjecture holds. This means that considering the complexity of decision, both parameterized and unparameterized, and of approximation independently is insufficient. Researchers and practicioners should consider the complexity of verification in addition to the complexity of decision and optimization.

This line of research, along with the fact that the complexity of solving a decision problem seems to have little to with the approximability or parameterized complexity of that problem, emphasizes the need to further determine the relative difficulty of computational problems with respect to the complexity of decision, verification, and approximation. Descriptive complexity seems like a promising way of unifying the complexity analyses of the different kinds of computational problems explored here; there are descriptive complexity characterizations for classes of decision problems [44], classes of approximable optimization problems [51], and classes of parameterized problems [35].

Chapter 2

Decision problems

One of the major successes of the PCP theorem, a characterization of NP as a class of computational problems that have probabilistically checkable proof systems (with polynomial-time verifiers), is that it provides a route to proving that approximating certain computationally intractable optimization problems is as difficult as solving them exactly. The growth of multiprocessor systems in both general purpose personal computing and large-scale big data computations highlights the urgency of proving the analagous inapproximability (or approximability) of inherently sequential optimization problems by highly parallel algorithms. However, there has been little theoretical work toward proving parallel inapproximability. Unfortunately, the techniques used to prove the original PCP theorem rely on the fact that NP can be interpreted as the class of languages for which there is an efficient verification procedure given a brief witness to language membership; no such obvious interpretation of P exists.

If we consider the P-complete problems to be tractable but inherently sequential and NC problems to be highly parallelizable, then our guiding question is whether

there is a probabilistically checkable proof (PCP) characterization of P with NC verifiers. Such a characterization would potentially provide a path to proving parallel inapproximability. Indeed, this question was already on the minds of researchers such as Luca Trevisan soon after the original proof of the PCP theorem.

> An intriguing question is whether the known non-approximability results for sequential algorithms can be improved when we restrict to NC algorithms (under the assumption that $\mathsf{P} \neq \mathsf{NC}$). A possible way may be to devise *probabilistic proof systems for* P more efficient that the currently known proof systems for NP. Such a result would have a great independent interest. However, it is not clear why proofs for P should be *easier to check* than proofs for NP (they only appear to be *easier to generate*). [66]

As a first step towards characterizing probabilistic proof systems for P, this chapter provides some initial structural complexity results for classes of probabilistically checkable proof systems for nondeterministic NC circuit families

Perhaps P has proof systems that are easy to check in NC, but this remains unclear. Instead, we consider proof systems for the class NNC[polylog], the class of languages decidable by NC circuit families augmented with a polylogarithmic number of nondeterministic gates. Other researchers such as Jonathan Buss and Judy Goldsmith have had similar questions about classes like this.

> The fundamental question remains whether there are problems in P that can be computed more quickly with limited nondeterminism than without it. [11]

We consider NNC[polylog] for two reasons. First, it is defined in such a way that it explicitly has short proof systems which are easy to verify in parallel, just as NP is

defined in such a way that it explicitly has short proof systems which are easy to verify efficiently. Second, it, like P, lies between NC and NP.

Although our original intention was to show something like NNC[polylog] = PCP[$O(\log \log n), O(1)$], our research reveals that proving such an equality is equivalent to proving NNC[polylog] = NC, or in other words, that a polylogarithmic amount of nondeterminism can be simulated deterministically by an NC circuit family. This should be seen as evidence that such a result is unlikely; in fact, we show that such a simulation implies a deterministic subexponential time algorithm for the Boolean formula satisfiability problem! We are still, however, able to show that certain PCP classes are contained in NNC[polylog].

2.1 History

In the 1990s, the PCP theorem by Arora, et al. [5], the culmination of a line of research attempting to trade nondeterminism for randomness in nondeterministic polynomial-time algorithms, provided a surprising new technique for verifying a mathematical proof: as long as the proof is converted to a certain format, an algorithm using a small amount of randomness can decide whether the proof is correct by examining a constant number of bits of the proof. On top of this fascinating illustration of the power of randomness in computation and mathematics, the PCP theorem provides a theoretical basis for proving inapproximability of NP optimization problems by polynomial-time algorithms. In 1991, Feige, et al. [33] used a generic gap-introducing reduction from a probabilistically checkable proof system to the maximum clique problem to show that the problem is hard to approximate by a polynomial-time algorithm. From there, any approximation-preserving reduction (see chapter 3) from the maximum clique problem to another optimization problem proves a similar level of inapproximability.

Other researchers examined different settings of parameters for the PCP theorem. For example, in 1996, Fotakis and Spirakis [36] provided a lower bound on the amount of randomness needed when creating a PCP verifier for an NP problem. In 1998, Trevisan examined inapproximability by parallel algorithms instead of inapproximability by polynomial-time algorithms for the linear programming problem [66].

The original proof of the PCP theorem is complicated and computational in nature. Inspired by some techniques for constructing explicit constructions of expander graphs like that of Reingold, Vadhan, and Wigderson [57], in 2007, Dinur provided a simpler (and almost entirely combinatorial) proof of the PCP theorem [28]. More recent work on proof systems has focused on practical real-world implementations in which the prover and the verifier have some limited communication; see articles by Goldwasser, Kalai, and Rothblum in 2008 [39], Setty et al. in 2012 [61], Setty et al. in 2012 [62], Thaler et al. in 2012 [64], and Ben-Sasson et al. in 2013 [9].

Around the same time as the PCP theorem, Wolf studied the class of nondeterministic highly parallel algorithms, NNC, and noted that a polylogarithmic amount of nondeterminism was an "interesting" amount of nondeterminism, suggesting that such a class may be incomparable with P [68]. The deterministic class NC, an abbreviation of *Nick's Class* in honor of Nick Pippenger, has been considered the class of decision problems that admit highly parallel algorithms since the 1970s; for a more detailed history of the study of parallel versus sequential computation, see [40, Section 1.3]. The complexity classes $\mathsf{NNC}^k[\log^i n]$ were proven to have complete problems by Cai and Chen in 1997 [15]. The study of limited nondeterminism for polynomial-time algorithms was initiated in 1980 by Kintala and Fisher [49], and advanced by several other researchers. Perhaps the most relevant to this dissertation are the articles by Díaz and Torán in 1990 [26], Buss and Goldsmith in 1993 [11], and Cai and Chen in 1997 [15].

2.2 Preliminaries

Throughout this chapter, $\log n$ denotes the base 2 logarithm of n. In the definitions below, $\mathbb{N}$ denotes the set of non-negative integers and $\mathbb{R}$ denotes the set of real numbers.

Definition 2.2.1. For all functions $f, g\colon \mathbb{N} \to \mathbb{R}$, the function f is in the class $O(g(n))$ if there exist real numbers c and N such that for all natural numbers n we have $n > N$ implies $f(n) \leq c \cdot g(n)$. If $f(n) < c \cdot g(n)$ then $f(n)$ is in $o(g(n))$. If $f(n) \geq c \cdot g(n)$ then $f(n)$ is in $\Omega(g(n))$. If $f(n) > c \cdot g(n)$ then $f(n)$ is in $\omega(g(n))$.

We assume the reader knows the basic definitions from complexity theory, including those of the complexity classes P, NP, DTIME, and DSPACE. We define the class L^k by $\mathsf{L}^k = \mathsf{DSPACE}(\log^k n)$ for all nonnegative integers k and the class polyL by $\mathsf{polyL} = \cup_{k\in\mathbb{N}}\mathsf{DSPACE}(\log^k n)$. We denote the class L^1 by simply L. We define the complexity class SUBEXP, the class of languages decidable by deterministic "subexponential" time Turing machines, as

$$\mathsf{SUBEXP} = \bigcap_{\epsilon>0} \mathsf{DTIME}(2^{n^\epsilon})$$

and QP, the class of languages decidable by a deterministic "quasipolynomial" time Turing machine, as

$$\mathsf{QP} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}(2^{\log^k n}).$$

We will also be considering NC^k, the class of languages decidable by a family of logarithmic space uniform Boolean circuits of polynomial size, $O(\log^k n)$ depth, and unbounded fan-in. We will denote by NC the union of all the NC^k classes. A language in NC^k can also be described as a language which admits an algorithm which uses a polynomial number of processors running in $O(\log^k n)$ time on a parallel random-access

machine (PRAM). We describe NC algorithms using this paradigm.

Definition 2.2.2. A *probabilistically checkable proof verifier* (*PCP verifier*) is a probabilistic Turing machine with sequential access to an input string x, sequential access to a random string ρ, and *nonadaptive random access* to a proof string π.

Definition 2.2.3. Let $r(n)$ and $q(n)$ be bounded above by polynomials in n, and let $c(n)$ and $s(n)$ be functions whose values are in the interval $[0, 1]$. A language L has a $(r(n), q(n), c(n), s(n))$-*PCP verifier* if there exists a PCP verifier V such that V uses at most $r(n)$ bits of the random string ρ, makes at most $q(n)$ *nonadaptive* queries to bits of the proof π, and satisfies the following conditions.

1. If $x \in L$, then

$$\exists \pi \in \Sigma^* : \Pr_{\rho \in \Sigma^{r(n)}} [V(x, \pi; \rho) \text{ accepts}] \geq c(n).$$

2. If $x \notin L$, then

$$\forall \pi \in \Sigma^* : \Pr_{\rho \in \Sigma^{r(n)}} [V(x, \pi; \rho) \text{ accepts}] < s(n).$$

The value $c(n)$ is the *completeness* and the value $s(n)$ the *soundness* of the verifier.

In this chapter, we will consider only nonadaptive PCP verifiers. Since a (nonadaptive) $(r(n), q(n), c(n), s(n))$-PCP verifier can read at most $2^{r(n)}q(n)$ locations of the proof string with nonzero probability, we assume without loss of generality that the proof provided to the verifier is of length at most $2^{r(n)}q(n)$ [3, Remark 11.6]. (Note that a verifier which uses $q(n)$ adaptive random access queries to the proof string can be simulated by a verifier which uses $2^{q(n)}$ nonadaptive random access queries to the proof string, so in the adaptive case, the proof string could be of length $2^{r(n)+q(n)}$.)

Definition 2.2.4. Let $\mathsf{PCP}^{\mathcal{C}}_{c(n),s(n)}[r(n),q(n)]$ be the class of all languages L such that L has a $(r(n),q(n),c(n),s(n))$-PCP verifier V computable by a $\mathcal{C}$ algorithm.

More generally, if $\mathcal{F}$ and $\mathcal{G}$ are classes of functions,

$$\mathsf{PCP}^{\mathcal{C}}_{c(n),s(n)}[\mathcal{F},\mathcal{G}] = \bigcup_{f\in\mathcal{F},g\in\mathcal{G}} \mathsf{PCP}^{\mathcal{C}}_{c(n),s(n)}[f(n),g(n)],$$

Since completeness 1 and soundness ½ are common parameters, and for the sake of brevity, we write $\mathsf{PCP}^{\mathcal{C}}[r(n),q(n)]$ to denote $\mathsf{PCP}^{\mathcal{C}}_{1,\frac{1}{2}}[r(n),q(n)]$, and $\mathsf{PCP}^{\mathcal{C}}[\mathcal{F},\mathcal{G}]$ to denote $\mathsf{PCP}^{\mathcal{C}}_{1,\frac{1}{2}}[\mathcal{F},\mathcal{G}]$.

Please notice that the complexity class given in the superscript in the above definition does *not* denote an oracle; it merely describes the computational power of the PCP verifier.

From the definition, we see immediately that

$$\mathsf{PCP}^{\mathcal{C}}_{c(n),s(n)}[O(r(n)),O(q(n))] = \bigcup_{a\in\mathbb{N},b\in\mathbb{N}} \mathsf{PCP}^{\mathcal{C}}_{c(n),s(n)}[a\cdot r(n),b\cdot q(n)].$$

2.3 Probabilistically checkable proofs for nondeterministic circuits

We first provide a PCP characterization of $\mathsf{NNC}[\mathsf{polylog}]$, then later we provide upper and lower bounds for the randomness and query complexity parameters of such a PCP verifier. The following theorem shows that a nondeterministic NC circuit can simulate a PCP verifier and vice versa with the appropriate tradeoff in parameters.

Theorem 2.3.1. *For all nonnegative integers q and r,*

$$\mathsf{NNC}[\log^q n] \subseteq \mathsf{PCP}^{\mathsf{NC}}[r\log\log n, O(\log^q n)] \subseteq \mathsf{NNC}[\log^{q+r} n].$$

Proof. Let q and r be non-negative integers. The first inclusion, $\mathsf{NNC}[\log^q n] \subseteq \mathsf{PCP}^{\mathsf{NC}}[r \log\log n, O(\log^q n)]$, follows immediately from the definitions. For the other direction, suppose $L \in \mathsf{PCP}^{\mathsf{NC}}[r \log\log n, c \log^q n]$ for some constant c. Construct an NNC machine M which proceeds as follows on input x of length n.

1. Guess a proof string π of length $2^{r \log\log n} c \log^q n$.
2. For each ρ of length $r \log\log n$ *in parallel* simulate $V(x, \pi; \rho)$.
3. Accept if and only if at least half of the simulations accept.

In the initial step, guessing a proof string requires $O(\log^{q+r} n)$ bits. In the second step, since V is an NC machine, a polylogarithmic number of parallel simulations of V can be executed with only a polylogarithmic factor increase in size and no increase in depth. In the final step, computing the majority of a polylogarithmic number of bits can be done by an NC circuit. Therefore M is an $\mathsf{NNC}[\log^{q+r} n]$ machine. The correctness of M follows from the completeness and soundness of V. □

Choosing $r = 1$ yields

$$\mathsf{NNC}[\log^q n] \subseteq \mathsf{PCP}^{\mathsf{NC}}[\log\log n, O(\log^q n)] \subseteq \mathsf{NNC}[\log^{q+1} n].$$

On the other hand, allowing r and q to vary over the set of natural numbers proves the equality of the two hierarchies.

Corollary 2.3.2. $\mathsf{NNC}[\mathsf{polylog}] = \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}]$.

Next, consider the chain of inclusions

$$\mathsf{NNC}[\log n] \subseteq \mathsf{NNC}[\mathsf{polylog}] \subseteq \mathsf{NNC}[\mathsf{poly}].$$

In fact, $\mathsf{NC} = \mathsf{NNC}[\log n]$ and $\mathsf{NNC}[\mathsf{poly}] = \mathsf{NP}$ [68], so we can rewrite this as

$$\mathsf{NC} \subseteq \mathsf{NNC}[\mathsf{polylog}] \subseteq \mathsf{NP}. \tag{2.1}$$

We now wish to provide NC PCP characterizations for both NC and NP.

In [36], the authors prove that $\mathsf{P} = \mathsf{PCP}^{\mathsf{P}}[O(\log\log n), O(1)]$ (implicitly; they state only that $\mathsf{NP} = \mathsf{PCP}^{\mathsf{P}}[O(\log\log n), O(1)]$ if and only if $\mathsf{P} = \mathsf{NP}$). The same proof techniques can be used in the NC setting with essentially no changes. (The idea of the proof is to simulate $O(\log\log n)$ bits of randomness with $\log\log n + O(1)$ bits by making a random walk of an appropriate length on a fully explicit constant degree expander graph.) This yields the following PCP characterization of NC.

Theorem 2.3.3. $\mathsf{NC} = \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(1)]$.

As a generalization of the result of [36], we know $\mathsf{NP} = \mathsf{PCP}^{\mathsf{P}}[o(\log n), o(\log n)]$ if and only if $\mathsf{P} = \mathsf{NP}$ [4, 33]. Unfortunately, the obvious strategy for translating that proof to the NC setting fails. The proof would have shown that $\mathsf{NP} = \mathsf{PCP}^{\mathsf{NC}^k}[o(\frac{\log\log n}{\log^k n}), O(1)]$ if and only if $\mathsf{NC} = \mathsf{NP}$, but this is already proven by Theorem 2.3.3 and the fact that $\frac{\log\log n}{\log^k n} \leq \log\log n$.

We also know the following strengthening of the original PCP theorem; the proof of this theorem is in section 2.5.

Theorem 2.3.4. $\mathsf{PCP}^{\mathsf{NC}}[O(\log n), O(1)] = \mathsf{NP}$.

From Equation 2.1, Theorem 2.3.3, and Theorem 2.3.4, we have the two equivalent inclusion chains $\mathsf{NC} \subseteq \mathsf{NNC}[\mathsf{polylog}] \subseteq \mathsf{NP}$ and

$$\mathsf{PCP}[O(\log\log n), O(1)] \subseteq \mathsf{PCP}[O(\log\log n), \mathsf{polylog}] \subseteq \mathsf{PCP}[O(\log n), O(1)],$$

where the PCP verifier is an NC machine. If we can provide evidence that $\mathsf{NC} \neq \mathsf{NNC}[\mathsf{polylog}]$ and that $\mathsf{NNC}[\mathsf{polylog}] \neq \mathsf{NP}$, we can conclude that the corresponding PCP classes are also likely distinct. This theorem, adapted from [26, Theorem 1] (therein attributed to R. Beigel), provides that evidence. Each of the two conclusions in this theorem implies that the exponential time hypothesis is false. Furthermore, in the latter case, the conclusion implies that $\mathsf{EXP} = \mathsf{NEXP}$.

Theorem 2.3.5.

1. *If* $\mathsf{NC} = \mathsf{NNC}[\mathsf{polylog}]$, *then* $\mathsf{NP} \subseteq \mathsf{SUBEXP}$.

2. *If* $\mathsf{NNC}[\mathsf{polylog}] = \mathsf{NP}$, *then* $\mathsf{NP} \subseteq \mathsf{QP}$.

Proof. If $\mathsf{NNC}[\mathsf{polylog}] = \mathsf{NP}$, then

$$\begin{aligned}
\mathsf{NP} &= \mathsf{NNC}[\mathsf{polylog}] \\
&\subseteq \mathsf{DSPACE}(\mathsf{polylog}) && \text{by [68]} \\
&\subseteq \mathsf{DTIME}(2^{\mathsf{polylog}}) && \text{by exhaustive search} \\
&= \mathsf{QP} && \text{by definition.}
\end{aligned}$$

Now suppose $\mathsf{NC} = \mathsf{NNC}[\mathsf{polylog}]$. Since FSAT, the Boolean formula satisfiability problem, is complete for NP under deterministic polynomial-time many-one reductions, it suffices to show a deterministic subexponential time algorithm for FSAT.

The proof uses a padding argument. First we observe that there is an $\mathsf{NNC}^1(n)$ machine, call it M, that decides FSAT: given a Boolean formula ϕ, guess a satisfying assignment to ϕ (of length $O(n)$) and evaluate the formula (Boolean formula evaluation is in NC^1 [12]). Let ϵ be an arbitrarily small positive constant, and define L, the

padded version of FSAT, as

$$L = \left\{ \phi \# 1^P \,\middle|\, \phi \in \text{FSAT and } P = 2^{n^\epsilon} - (n+1) \right\},$$

where $n = |\phi|$. We claim L is in $\mathsf{NNC}[\log^{\frac{1}{\epsilon}} n]$ by the following machine, M_j. On input ϕ', check that ϕ' is in the format $\phi \# 1^P$, then accept if and only if M accepts ϕ. The correctness of this algorithm follows from the correctness of M, so it remains to check the size and depth of the circuit for M_j, and the amount of nondeterminism used.

Checking that x' is in the correct format can be performed (deterministically) by an NC^1 circuit by computing the conjunction of all the bits after the # symbol. Observe now that $|x'| = 2^{n^\epsilon}$, so $n = \log^{\frac{1}{\epsilon}} |x'|$. The amount of nondeterminism used by M_j is the same as the amount used by M, which is $O(n)$, or $O(\log^{\frac{1}{\epsilon}} |x'|)$. The size of M is polynomial in n, which is polylogarithmic in $|x'|$, and hence polynomial in the length of the input x'. The depth of M is $O(\log n)$, which is $O(\log \log^{\frac{1}{\epsilon}} |x'|)$, or simply $O(\log \log |x'|)$. We conclude that the size of M_j is polynomial in $|x'|$, the depth of M_j is logarithmic in $|x'|$, and M_j uses $O(\log^{\frac{1}{\epsilon}} |x'|)$ bits of nondeterminism. Hence L is in $\mathsf{NNC}[\log^{\frac{1}{\epsilon}} n]$.

By hypothesis, L is also in NC. Let M_i be the NC machine that decides it. We claim that we can now construct a subexponential time algorithm for FSAT on inputs ϕ of length n.

1. Let $\phi' = \phi \# 1^P$, where $P = 2^{n^\epsilon} - (n+1)$.

2. Accept if and only if M_i accepts ϕ'.

The correctness of this algorithm follows immediately from the correctness of M_i. The first step can be performed by a deterministic algorithm running in time 2^{n^ϵ}. The second step can be performed by an NC machine. Since $\mathsf{NC} \subseteq \mathsf{P}$, and 2^{n^ϵ} is greater

than any polynomial for sufficiently large n, the first step is the bottleneck in this algorithm. Therefore, this algorithm for FSAT can be implemented by a deterministic algorithm running in $O(2^{n^\epsilon})$ time for arbitrarily small ϵ. □

As mentioned above, $\mathsf{NC} = \mathsf{NNC}[\log n]$ and $\mathsf{NP} = \mathsf{NNC}[\mathsf{poly}]$, so the two items in this theorem can be restated as follows.

1. If $\mathsf{NNC}[\log n] = \mathsf{NNC}[\mathsf{polylog}]$, then $\mathsf{NP} \subseteq \mathsf{SUBEXP}$.
2. If $\mathsf{NNC}[\mathsf{polylog}] = \mathsf{NNC}[\mathsf{poly}]$, then $\mathsf{NP} \subseteq \mathsf{QP}$.

The first item indicates that a simulation of a polylogarithmic number of bits of nondeterminism by only a logarithmic number of bits is unlikely. The second item indicates that a simulation of a polynomial number of bits of nondeterminism by only a polylogarithmic number of bits is unlikely. The consequences of the latter simulation are more extreme (a simulation of NP in quasipolynomial time as opposed to a simulation of NP in subexponential time).

Substituting the PCP characterizations of each nondeterministic NC complexity class in the previous theorem provides evidence against the simulation of certain resources in probabilistically checkable proof systems.

Corollary 2.3.6.

1. *If* $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(1)] = \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}]$, *then* $\mathsf{NP} \subseteq \mathsf{SUBEXP}$.
2. *If* $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}] = \mathsf{PCP}^{\mathsf{NC}}[O(\log n), O(1)]$, *then* $\mathsf{NP} \subseteq \mathsf{QP}$.

The first part of this corollary provides evidence that for certain classes of computational problems, an NC PCP verifier cannot reduce the number of necessary queries. (However, it could still be the case that for some fixed positive integer k, we

have $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(1)] = \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(\log^k n)]$; see Conjecture 2.3.7 below.) The second part provides evidence that for certain classes of computational problems, a verifier cannot reduce randomness in exchange for an increase in the number of necessary queries. Contrast this with [36, Corollary 10] which states that

$$\mathsf{PCP}^{\mathsf{NC}}[O(\log^k \log n), O(\log^d \log n)] \subseteq \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(\log^{d+k-1} \log n)]$$

(the result is proven for polynomial-time verifiers, but it holds for NC verifiers as well). This yields the equality

$$\mathsf{PCP}^{\mathsf{NC}}[\mathsf{poly}(\log\log n), \mathsf{poly}(\log\log n)] = \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{poly}(\log\log n)],$$

which provides an even more severe collapse, assuming the following conjecture.

Conjecture 2.3.7. $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(\log n)] = \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(1)]$.

This is a scaled down version (that is, scaled from a P verifier down to an NC verifier) of some of the results of the research which led to the original PCP theorem. If this conjecture holds, then $\mathsf{poly}(\log\log n)$ randomness and a logarithmic number of queries to the proof can be simulated deterministically.

Theorem 2.3.8. *If Conjecture 2.3.7 holds, then* $\mathsf{PCP}^{\mathsf{NC}}[\mathsf{poly}(\log\log n), O(\log n)] = \mathsf{NC}$.

Proof. Combining Conjecture 2.3.7 with the fact that $O(\log^\alpha \log n) \subseteq O(\log n)$ for all

nonnegative integers α, we have

$$\begin{aligned}
\mathsf{NC} &\subseteq \mathsf{PCP}^{\mathsf{NC}}[\mathsf{poly}(\log\log n), \mathsf{poly}(\log\log n)] \\
&\subseteq \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{poly}(\log\log n)] \\
&\subseteq \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(\log n)] \\
&\subseteq \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), O(1)] \\
&\subseteq \mathsf{NC}.
\end{aligned}$$

□

Now we return to our original goal, finding a PCP characterization of P. The classes P and NNC[polylog] are conjectured incomparable [68]. Using the results above, this conjecture implies that P and $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}]$ are incomparable. Theorem 2.3.9 shows the negative consequences of a PCP characterization for P.

Theorem 2.3.9.

1. *If* $\mathsf{P} \subseteq \mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}]$ *then* $\mathsf{P} \subsetneq \mathsf{polyL}$.
2. *If* $\mathsf{polyL} \subsetneq \mathsf{P}$ *then* $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}] \subsetneq \mathsf{P}$.

Proof. These implications are a consequence of three facts.

1. $\mathsf{PCP}^{\mathsf{NC}}[O(\log\log n), \mathsf{polylog}] \subseteq \mathsf{NNC}[\mathsf{polylog}]$ (Corollary 2.3.2).
2. $\mathsf{NNC}[\mathsf{polylog}] \subseteq \mathsf{polyL}$ ([68, Corollary 3.2]).
3. $\mathsf{P} \neq \mathsf{polyL}$ ([10, Theorem 3.10]). □

Although $\mathsf{P} \neq \mathsf{polyL}$, whether one is a strict subset of the other remains unknown; the two are conjectured to be incomparable [45, Section 2.5.1]. If $\mathsf{P} \subsetneq \mathsf{polyL}$, then $\mathsf{P} \subsetneq \mathsf{QP}$ (by exhaustive search over the quasipolynomial number of configurations of the polyL machine). If $\mathsf{polyL} \subsetneq \mathsf{P}$, then $\mathsf{L} \subsetneq \mathsf{L}^2 \subsetneq \cdots \subsetneq \mathsf{polyL} \subsetneq \mathsf{P}$.

2.4 Inapproximability from PCPs

Consider the *maximum high degree subgraph problem* ([2]): given an undirected graph G find the largest integer d such that G has a vertex-induced subgraph of minimum degree d. This is a relaxation of the maximum clique problem, in which the minimum degree of the induced subgraph S is required to be at least $|S| - 1$. There is a simple polynomial-time algorithm that outputs optimal solutions for this problem: repeatedly remove vertices of degree less than d from the graph. The subgraph that remains has minimum degree d; for more information, see [40, Problem A.2.7].

We know that the clique problem is inapproximable via a gap-introducing reduction from an arbitrary probabilistically checkable proof system [33]. It would be satisfying to use a similar reduction to provide a gap-introducing reduction from our restricted PCPs to the maximum high degree subgraph problem. However, research in this direction failed to reveal such a reduction.

2.5 Probabilistically checkable proofs for nondeterministic polynomial time

One method of showing $\mathsf{PCP}^{\mathsf{NC}}[O(\log n), O(1)] = \mathsf{NP}$ is to revisit a proof of the PCP theorem and ensure that all computation can be performed by an NC PCP verifier without affecting the correctness of the proof. We will consider Dinur's proof of the PCP theorem [28], which reduces the problem of proving $\mathsf{PCP}^{\mathsf{P}}[O(\log n), O(1)] = \mathsf{NP}$ to the problem of showing $\frac{1}{2}$-GAP q-CSP is hard for NP. Meir observes that although we would like a verifier running in polylogarithmic time, Dinur's proof requires $O(\log n)$ iterations of a polynomial-time procedure, which yields a polynomial-time procedure [54, Section 1.2.1]. We show that a closer look reveals that parallel polylogarithmic time is indeed possible without any new machinery.

The proof provides a gap-introducing reduction from an arbitrary NP problem to a constraint satisfaction problem. Let us first define the notion of a combinatorial constraint and when a constraint is satisfied.

Definition 2.5.1 ([28, Definition 1.1])**.** Let V be a finite set of variables, defined by $V = \{x_1, x_2, \ldots, x_n\}$, let Γ be a finite alphabet, and let q be a natural number. A *q-ary constraint* is a $q+1$ tuple, $(C, i_1, i_2, \ldots, i_q)$, where $C \subseteq \Gamma^q$ and each i_k is the index of a variable in V. Here, C is considered the set of "acceptable" values for the variables and each i_k is the index of a variable whose assigned value will be checked against C.

An *assignment* is a function $a \colon V \to \Gamma$. An assignment *satisfies* a constraint if $(a(v_{i_1}), a(v_{i_2}), \ldots, a(v_{i_q})) \in C$.

A natural question for a given set of constraints is whether there is an assignment to the variables that simultaneously satisfies all the constraints. A related problem asks the same question but given the guarantee that either all the constraints are satisfied or few of the constraints are satisfied, regardless of the assignment.

Definition 2.5.2 (q-CSP)**.**

Instance: finite alphabet Γ with $|\Gamma| > 1$, finite set of variables V, finite set of q-ary constraints D.

Question: Are all constraints in D satisfiable?

Definition 2.5.3 ($\frac{1}{2}$-GAP q-CSP)**.**

Instance: finite alphabet Γ with $|\Gamma| > 1$, finite set of variables V, finite set of q-ary constraints D with the restriction that for any assignment, either all constraints are satisfied or fewer than half are.

Question: Are all constraints in D satisfiable?

In the special case in which $q = 2$, that is, all constraints are binary, we may interpret an instance of the constraint satisfaction problem as an undirected graph with vertex set V and an edge labeled C between vertices v_i and v_j for each (C, i, j) in D. We call such a graph a *constraint graph* and we consider the size of this graph to be $|V| + |E|$ where V is the set of vertices (equivalently, variables) and E is the set of edges.

Lemma 2.5.4. *If there is a positive integer q such that $\frac{1}{2}$-GAP q-CSP is hard for* NP *under* NC *many-one reductions, then* $\mathsf{PCP}^{\mathsf{NC}}[O(\log n), O(1)] = \mathsf{NP}$.

Proof. One inclusion in the conclusion of the theorem is true unconditionally, following from the PCP theorem [5]. For the other inclusion, let L be a language in NP. By hypothesis there is a many-one reduction computable in NC from L to $\frac{1}{2}$-GAP q-CSP. We construct the PCP verifier as follows.

1. Compute the reduction to produce a set of constraints.
2. Use $O(\log n)$ random bits to choose a constraint uniformly at random.
3. Check that the constraint is satisfied by querying the proof string at the appropriate locations (the locations corresponding to the q variables in the constraint).

The first step is computable in NC by hypothesis. The second step uses $O(\log n)$ bits of randomness and a constant number of parallel steps. In the third step, q processors, working in parallel, each read the index of a variable given in the chosen constraint, then retrieve the value of that variable given in the proof string; this takes at most $O(\log n)$ parallel time. Verifying that the constraint is satisfied by these values is just the problem of checking q set membership queries in parallel, which again can be done in $O(\log n)$ time (a loose upper bound). The overall number of processors in

this algorithm is polynomial and the overall parallel time is polylogarithmic, thus this an NC algorithm.

It remains to show that the algorithm is correct. If $x \in L$ then all constraints are satisfiable, so there exists an assignment such that the verifier will accept on all random choices of the constraint. If $x \notin L$ then fewer than half of the constraints are satisfiable, so for any assignment the probability that the verifier will select a satisfied constraint is less than half. Therefore we have shown a correct PCP verifier with the appropriate parameters for an arbitrary language in NP. □

Now we examine Dinur's proof that $\frac{1}{2}$-GAP q-CSP is hard for NP [28]. That proof shows that the problem is hard under polynomial-time many-one reductions, but we show here that it is in fact hard under NC many-one reductions. First, we claim without proof that q-CSP is hard for NP under NC many-one reductions (because the standard polynomial-time many-one reductions showing that it is NP-complete are in fact computable in logarithmic space). Next, consider (the high-level description of) the polynomial-time many-one reduction from q-CSP to $\frac{1}{2}$-GAP q-CSP: given constraint graph G_0 as input, compute and output $G_{O(\log n)}$, where $G_{i+1} = \mathcal{P}((X(G_i))^t)$ and $t \in O(1)$. Here, X is a preprocessing function, the exponent t denotes a constant number of constraint graph powering operations, and $\mathcal{P}$ denotes an assignment testing composition function. If each of these three functions is computable by an NC algorithm, then G_{i+1} can be computed from G_i by an NC algorithm, and hence so can $G_{O(\log n)}$ from G_0. We will consider each of the three functions below.

The preprocessing function X requires a *mildly explicit* construction of a constant degree expander. The standard definition of mildly explicit is that a representation of the graph (for example, its adjacency matrix) is computable in time polynomial in the number of vertices in the graph; for comparison, in a *fully explicit* expander, the

ith neighbor of vertex v can be computed in time polynomial in the size of the binary representation of v, that is, polynomial in $\log n$ where n is the number of nodes in the graph. We will refer to graphs which meet these definitions as *polynomial-time mildly explicit* and *polynomial-time fully explicit*. Since we are constructing an NC algorithm, we will require the representation of the graph to be computable by an NC algorithm. More formally, we require an NC *mildly explicit* graph, that is, a graph for which a representation can be computed by an NC algorithm with respect to input n, the number of nodes of the graph. Fortunately, a polynomial-time fully explicit expander implies an NC mildly explicit expander, a new implication that may be of independent interest when constructing constant-degree expanders in parallel.

Proposition 2.5.5. *Suppose G is a d-regular expander graph. If G is polynomial-time fully explicit, then it is* NC *mildly explicit.*

Proof. Suppose G is polynomial-time fully explicit, so there exists an algorithm that computes the ith neighbor of v in time polynomial in $\log n$. Let $f(v, i)$ denote this algorithm. The following NC algorithm computes the adjacency list of G given the number of nodes n: for each vertex v *in parallel* and each index i less than d *in parallel* add $f(v, i)$ to the list corresponding to v. This algorithm can be computed with dn processors, which is polynomial in n. Since the $f(v, i)$ can be computed in time polynomial in $\log n$, the running time for each parallel processor is also polynomial in $\log n$. Therefore we have presented an NC algorithm which correctly computes a representation of the graph G. □

This proposition allows us to replace any polynomial-time fully explicit expander with an NC mildly explicit one. Dinur's proof only requires only polynomial-time mildly explicit expanders, but replacing that requirement with fully explicit ones

harms neither the correctness nor the efficiency of the construction. Polynomial-time fully explicit constant degree expander graphs exist; see [57], for example.

Now, let us return to the preprocessing function X, which is defined in two parts, [28, Definition 4.1] and [28, Definition 4.2]. In the first part, each vertex v is replaced by an NC mildly explicit d-regular expander on $\deg(v)$ vertices in which the constraints on the edges of the expander are the equality constraint. In the second part, a constant number of self-loops along with the edges of a d'-regular NC mildly explicit expander on n vertices are added to the graph with null constraints on the added edges. Both of these parts are computable by an NC algorithm; note that the size of the output graph in each case is linear in the size of the input graph, so a linear number of processors will suffice (with an additional multiplicative factor of a polynomial number of processors when constructing the expander graphs). We conclude the following.

Lemma 2.5.6. *The preprocessing function X is computable by an* NC *algorithm.*

Constraint graph powering, defined in [28, Section 1.2], is the standard graph powering operation with an additional operation on the alphabet and the set of constraints. Graph powering can be viewed as a generalization of computing the transitive closure of a graph; it computes not only whether there is a path joining two nodes but also the number of paths joining them. Graph powering can be performed by computing the appropriate power of the adjacency matrix of the graph. This can be computed in NC because matrix multiplication is in NC, and a constant number of matrix multiplications remains in NC. If the alphabet is of constant size, the power graph also has an alphabet of constant size, and each of the constraints becomes a new constraint of constant size (see [28, Section 1.2] for the constraint construction). Each of the constraints (which is the same as the number of edges) can be written by a distinct processor in parallel constant time. We conclude the following.

Lemma 2.5.7. *For each positive integer k and each positive integer d, computing the kth power of a d-regular constraint graph can be performed by an* NC *algorithm.*

The assignment testing composition [28, Definition 5.1] consists of two parts. In the first part, each constraint is transformed into a Boolean circuit of constant size. In the second part, each circuit constructed in this way is provided as input to a computable assignment tester function (which we know exists [28, Theorem 5.1]), and the output graph is the union of the output of all the assignment testers. Since the size of the input to the assignment tester is constant, the assignment tester need only be computable. Hence, each constraint can be processed this way, in parallel, in constant time with respect to the size of the input graph.

Lemma 2.5.8. *The assignment testing composition is computable by an* NC *algorithm.*

Since the preprocessing function, constraint graph powering, and assignment testing composition are all computable by an NC algorithm, we conclude the following.

Lemma 2.5.9. *There is a positive integer q such that $\frac{1}{2}$-GAP q-CSP is hard for* NP *under* NC *many-one reductions.*

Theorem 2.3.4, restated here, follows immediately from Lemma 2.5.4 and the hardness of the $\frac{1}{2}$-GAP q-CSP problem Lemma 2.5.9. Therefore we have shown that any decision problem with a polynomial-time verification procedure can be transformed into a probabilistically checkable proof system with a highly parallel verifier using a small amount of randomness and constant query complexity.

Theorem 2.3.4. $\mathsf{PCP}^{\mathsf{NC}}[O(\log n), O(1)] = \mathsf{NP}$.

Chapter 3

Optimization problems

Many natural computational problems can be expressed as optimization problems, allowing for a more refined analysis of the computational complexity of the problem. Much research has focused on efficient approximations of intractable optimization problems, with little work done to understand highly parallel approximations for tractable but otherwise inherently sequential optimization problems. Our task is to define the complexity classes associated with this notion and determine whether there are inherently sequential optimization problems that admit parallel approximations as well as whether there are sequential problems for which no parallel approximation exists. This chapter provides these complexity theoretic foundations.

Under reasonable complexity-theoretic assumptions, we prove that $\mathsf{NNCO} \subsetneq \mathsf{NPO}$ (Theorem 3.3.5) and $\mathsf{PO} \cap \mathsf{NNCO} \subsetneq \mathsf{PO}$ (Theorem 3.4.7). Furthermore, we prove that the hierarchies of classes of optimization problems approximable in parallel are strict (Theorem 3.5.2 and Theorem 3.5.3). Finally, we propose three candidate complete problems that we conjecture complete for the class of efficiently solvable but

constant-factor parallel approximable optimization problems.

These findings provide evidence that viewing computational problems through the lens of optimization provides a finer-grained understanding of their complexity, since $\mathsf{NNC}[\mathsf{poly}] = \mathsf{NP}$ but $\mathsf{NNCO} \subsetneq \mathsf{NPO}$ (under the appropriate assumptions). Also, the strictness of the hierarchy intersecting PO demonstrates that there are efficiently solvable problems of various levels of parallel approximability. Analagous to the hardness of approximation results for intractable problems, in some cases even approximating a solution is inherently sequential.

Altogether, this chapter reinforces the idea that the complexity of verifying a solution is an important factor in consider the overall computational complexity of an optimization problem.

3.1 History

The study of the computational complexity of NP optimization problems has existed since at least the early 1970s with Johnson [46] giving the first definitions of polynomial-time approximation algorithms. The definitions we use are inspired by those in the 1999 book *Complexity and Approximation* by Ausiello et al., which contains more detailed notes on the history of the computational complexity of NP optimization problems. Our study of optimization problems approximable by algorithms more restrictive than polynomial-time algorithms is guided by a 2007 article by Tantau [63] in which the author defines approximability and completeness for logarithmic-space optimization problems. NC approximations for NP-hard optimization problems have been studied by Hunt et al. in 1998 [43]. Although a class called NCX has been used to represent the class of optimization problems with NC constant-factor approximation algorithms, for example in a 1995 article by Serna and Xhafa [59] or the 1997 book

Paradigms for fast parallel approximability by Díaz et al. [27], its implicit definition requires only that the feasibility of a solution can be verified in polynomial time. This differs from our definition of ApxNCO, which requires that the solution be verifiable in NC.

After the definition of complexity classes based on approximability of NP optimization problems, the natural next task was to define completeness, which requires an appropriate notion of reducibility. A 1997 survey paper by Crescenzi [21] defines at least nine distinct types of polynomial-time approximation-preserving reductions, each of which serves its own purpose in relating the approximability of optimization problems. Articles by Ausiello, D'Atri, and Protasi [6] and Orponen and Manila [55] provided the first proof that the maximum weighted satisfiability problem is complete for NPO; this guides our proofs in section 3.3 and section 3.4 below. The first proof of an APX-complete problem comes from Crescenzi and Protasi in 1991 [23] and a proof for more natural complete problems come from Khanna et al. in 1999 [48] and Crescenzi and Trevisan in 2000 [22].

3.2 Definitions

Throughout this chapter, $\Sigma = \{0, 1\}$ and inputs and outputs are encoded in binary. The set of all finite strings is denoted Σ^*, and for each $x \in \Sigma^*$, we denote the length of x by $|x|$. We denote the set of all polynomials by poly and the set of all polylogarithmic functions by polylog. The set of integers is denoted $\mathbb{Z}$, the set of rationals $\mathbb{Q}$, and their positive subsets $\mathbb{Z}^+$ and $\mathbb{Q}^+$. The natural numbers, defined as $\mathbb{Z}^+ \cup \{0\}$, is denoted $\mathbb{N}$. Vectors are formatted in bold face, like $\mathbf{x}$. The all-ones vector is denoted $\mathbf{1}$.

3.2.1 Optimization problems and approximation algorithms

We adapt the definitions of [63] from L approximability to NC approximability.

Definition 3.2.1 ([7])**.** An *optimization problem* is a four-tuple, (I, S, m, t), where the set $I \subseteq \Sigma^*$ is called the *instance set*, the set $S \subseteq I \times \Sigma^*$ is called the *solution relation*, the function $m\colon S \to \mathbb{Z}^+$ is called the *measure function*, and $t \in \{\min, \max\}$ is called the *type* of the optimization.

An optimization problem in which the measure function has rational values can be transformed into one in which the measure function has integer values [7, page 23].

Definition 3.2.2 ([63])**.** Let P be an optimization problem, so $P = (I, S, m, t)$, and let $x \in I$.

1. Let $S(x) = \{y \in \Sigma^* \mid (x, y) \in S\}$; we call this the *solutions for* x.

2. Define $m^*(x)$ by

$$m^*(x) = \begin{cases} \min \{m(x, y) \mid y \in S(x)\} & \text{if } t = \min \\ \max \{m(x, y) \mid y \in S(x)\} & \text{if } t = \max \end{cases}$$

   for all $x \in \Sigma^*$; we call this the *optimal measure for* x. Let $m^*(x)$ be undefined if $S(x) = \emptyset$.

3. Let $S^*(x) = \{y \in \Sigma^* \mid m(x, y) = m^*(x)\}$; we call this the *set of optimal solutions for* x.

4. Let $R(x, y) = \max\left(\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\right)$; we call this the *performance ratio of the solution* y.

5. Let $P_\exists = \{x \in \Sigma^* \mid S(x) \neq \emptyset\}$; we call this the *existence problem*.

6. Let

$$P_{opt<} = \{(x, z) \in P_\exists \times \mathbb{N} \mid \exists y \in \Sigma^* \colon m(x, y) < z\}$$

and

$$P_{opt>} = \{(x, z) \in P_\exists \times \mathbb{N} \mid \exists y \in \Sigma^* \colon m(x, y) > z\};$$

we call these the *budget problems*.

7. Let $f \colon \Sigma^* \to \Sigma^*$. We say f *produces solutions for* P if for all $x \in P_\exists$ we have $f(x) \in S(x)$. We say f *produces optimal solutions for* P if for all $x \in P_\exists$ we have $f(x) \in S^*(x)$.

The performance ratio $R(x, y)$ is a number in the interval $[1, \infty)$. The closer $R(x, y)$ is to 1, the better the solution y is for x, and the closer $R(x, y)$ to ∞, the worse the solution.

Definition 3.2.3. Let P be an optimization problem, let $r \colon \mathbb{N} \to \mathbb{Q}^+$, and let $f \colon I \to \Sigma^*$. We say f is an *r-approximator for P* if it produces solutions for P and $R(x, f(x)) \leq r(|x|)$ for all $x \in P_\exists$.

If r is the constant function with value δ, we simply say f is a *δ-approximator for P*.

Definition 3.2.4. Let P be an optimization problem and let $f \colon I \times \mathbb{N} \to \Sigma^*$. We say f is an *approximation scheme for P* if for all $x \in P_\exists$ and all positive integers k we have $f(x, k) \in S(x)$ and $R(x, f(x, k)) \leq 1 + \frac{1}{k}$.

3.2.2 Classes of optimization problems

The study of *efficient* approximations for *intractable* problems begins with the following definition of NP optimization problems. We will adapt this definition to explore *efficient and highly parallel* approximations for *inherently sequential* problems.

Definition 3.2.5. The complexity class NPO is the class of all optimization problems (I, S, m, t) such that the following conditions hold.

1. The instance set I is decidable by a deterministic polynomial-time Turing machine.

2. The solution relation S is decidable by a deterministic polynomial-time Turing machine and is polynomially bounded (that is, the length of y is bounded by a polynomial in the length of x for all $(x, y) \in S$).

3. The measure function m is computable by a deterministic polynomial-time Turing machine.

The second condition is the most important in this definition; it is the analog of polynomial-time verifiability in NP.

Definition 3.2.6. The complexity class PO is the subclass of NPO in which for each optimization problem P there exists a function f in FP that produces optimal solutions for P.

We now wish to translate these definitions to the setting of efficient and highly parallel verifiability. In order to take advantage of results and techniques from the study of NPO and PO, we will start by considering a model of computation in which we allow highly parallel computation access to a polynomial amount of nondeterminism. First we define the necessary circuit classes, then we define the corresponding classes of optimization problems.

Definition 3.2.7.

1. NC is the class of decision problems decidable by a logarithmic space uniform family of Boolean circuits with polynomial size, polylogarithmic depth, and fan-in two.

2. FNC is the class of functions f computable by an NC circuit in which the output of the circuit is (the binary encoding of) $f(x)$.

3. $\mathsf{NNC}[f(n)]$ is the class of languages computable by a logarithmic space uniform NC circuit family augmented with $O(f(n))$ nondeterministic gates for each input length n [68]. A nondeterministic gate takes no inputs and yields a single (nondeterministic) output bit.

   If $\mathcal{F}$ is a class of functions, then $\mathsf{NNC}[\mathcal{F}] = \bigcup_{f \in \mathcal{F}} \mathsf{NNC}[f(n)]$.

$\mathsf{NNC}[\mathsf{poly}]$, also known as $\mathsf{GC}(\mathsf{poly}, \mathsf{NC})$ [15] , is an unusual class which may warrant some further explanation. NC has the same relationship to $\mathsf{NNC}[\mathsf{poly}]$ as P does to NP (thus an equivalent definition of $\mathsf{NNC}[\mathsf{poly}]$ is one in which each language has an efficient and highly parallel verification procedure; as in the definition of NPO in Definition 3.2.5, it is this formulation which we use when defining NNCO in Definition 3.2.8). Wolf [68] notes that $\mathsf{NNC}[\log n] = \mathsf{NC}$ and $\mathsf{NNC}[\mathsf{poly}] = \mathsf{NP}$, and suggests that $\mathsf{NNC}[\mathsf{polylog}]$ may be an interesting intermediary class, possibly incomparable with P. Cai and Chen [15] prove that for each natural number k and i, there is a complete problem for $\mathsf{NNC}^k \left[\log^i n\right]$ under logarithmic space many-one reductions.

Definition 3.2.8. The complexity class $\mathsf{NNCO}[\mathsf{poly}]$ is the class of all optimization problems (I, S, m, t) such that the following conditions hold.

1. The instance set I is decidable by an NC circuit family.

2. The solution relation S is decidable by an NC circuit family and is polynomially bounded (that is, the length of y is bounded by a polynomial in the length of x for all $(x, y) \in S$).

3. The measure function m is computable by an FNC circuit family.

For the sake of brevity, we write NNCO instead of NNCO[poly].

We can now proceed to define classes of approximable optimization problems contained in NNCO. Our guide for these definitions is the hierarchy of polynomial-time approximation classes between NPO and PO, namely APX, PTAS, and FPTAS.

Definition 3.2.9. Suppose P is an optimization problem in NNCO.

1. $P \in$ ApxNCO if there is an r-approximator in FNC for P, where $r(n) \in O(1)$ for all $n \in \mathbb{N}$.

2. $P \in$ NCAS if there is an approximation scheme f for P such that $f_k \in$ FNC for each $k \in \mathbb{N}$, where $f_k(x) = f(x, k)$ for all $x \in \Sigma^*$.

3. $P \in$ FNCAS if there is an approximation scheme f for P such that $f \in$ FNC in the sense that the size of the circuit is polynomial in both $|x|$ and k and the depth of the circuit is polylogarithmic in both $|x|$ and k.

4. $P \in$ NCO if there is a function f in FNC that produces optimal solutions for P.

For the NC approximation classes defined above, it is crucial that the solution relation is verifiable in NC. In all previous works (for example, [27, 59]), the implicit definition of, say, NCX, which corresponds to our class ApxNCO, requires only that the solution relation is verifiable *in polynomial time*. This important distinction does not seem to have been addressed before.

Each of the classes in Definition 3.2.9 includes the one defined below it. This chain of inclusions provides a hierarchy that classifies approximability of problems in NNCO, and hence in NPO,

$$\text{NCO} \subseteq \text{FNCAS} \subseteq \text{NCAS} \subseteq \text{ApxNCO} \subseteq \text{NNCO}.$$

However, our intention is to determine the approximability of optimization problems corresponding to P-complete decision problems, not those corresponding to NP-complete decision problems. Therefore we consider the classes $\mathsf{PO} \cap \mathsf{NNCO}$, $\mathsf{PO} \cap \mathsf{ApxNCO}$, etc. in order to more accurately capture the notion of highly parallel approximability of inherently sequential problems,

$$\mathsf{NCO} \subseteq \mathsf{PO} \cap \mathsf{FNCAS} \subseteq \mathsf{PO} \cap \mathsf{NCAS} \subseteq \mathsf{PO} \cap \mathsf{ApxNCO} \subseteq \mathsf{PO}.$$

The instance set, solution relation, and measure function of optimization problems in these classes are computable in NC, and furthermore, there is a polynomial-time algorithm that produces optimal solutions.

3.2.3 Reductions among approximation problems

There are many reductions for approximation problems; nine of them are defined in a survey paper by Crescenzi [21], and there are more defined elsewhere. We will use a logarithmic space-bounded version of the "AP reduction," considered by approximation experts to be a reasonable reduction to use when constructing complete problems [21, Section 2] [7, Section 8.6]. Although the original definition is from [25, Definition 9] (a preliminary version of [24, Definition 2.5]), the definition here is from [7, Definition 8.3].

Definition 3.2.10. [7, Definition 8.3] Let P and Q be optimization problems in NNCO, with $P = (I_P, S_P, m_p, t_P)$ and $Q = (I_Q, S_Q, m_Q, t_Q)$. We say P *AP reduces to* Q and write $P \leq_{AP}^{L} Q$ if there are functions f and g and a constant $\alpha \in \mathbb{R} \cap [1, \infty)$ such that

1. for all $x \in I_P$ and all $r \in \mathbb{Q} \cap (1, \infty)$, we have $f(x, r) \in I_Q$,

2. for all $x \in I_P$ and all $r \in \mathbb{Q} \cap (1, \infty)$, if $S_P(x) \neq \emptyset$ then $S_Q(f(x,r)) \neq \emptyset$,

3. for all $x \in I_P$, all $r \in \mathbb{Q} \cap (1, \infty)$, and all $y \in S_Q(f(x,r))$, we have $g(x,y,r) \in S_P(x)$,

4. f and g are computable in logarithmic space for any fixed r, and

5. for all $x \in I_P$, all $r > 1$, and all $y \in S_Q(f(x,r))$,

$$R_Q(f(x,r), y) \leq r \implies R_P(x, g(x,y,r)) \leq 1 + \alpha(r-1).$$

For a class $\mathcal{C}$ of optimization problems, we say a problem Q is *hard for* $\mathcal{C}$ if for all problems P in $\mathcal{C}$ there is a logarithmic space AP reduction from P to Q. If furthermore Q is in $\mathcal{C}$ we say Q is *complete for* $\mathcal{C}$.

3.3 Completeness in classes of inapproximable problems

This section shows that Maximum Variable-Weighted Satisfiability is complete for NNCO and Maximum Weighted Circuit Satisfiability is complete for NPO. Furthermore, the latter problem is not in NNCO unless NC = P. Thus there are optimization problems whose corresponding budget problems are of equal computational complexity—they are both NP-complete—but whose solution relations are of different computational complexity, under reasonable complexity theoretic assumptions.

The difference in the verification complexity between circuit and formula satisfiability problems appears also in chapter 4; compare these optimization problems with the parameterized problems p-FSat and p-CSat.

Definition 3.3.1 (Maximum Variable-Weighted Satisfiability)**.**

Instance: Boolean formula ϕ on variables $x_1, \ldots, x_n$, weights in $\mathbb{Q}^+$ for each variable $w_1, \ldots, w_n$.

Solution: assignment α to the variables that satisfies ϕ.

Measure: $\max(1, \Sigma_{i=1}^{n} \alpha(x_i) w_i)$.

Type: maximization.

Definition 3.3.2 (Maximum Weighted Circuit Satisfiability)**.**

Instance: Boolean circuit C with inputs $x_1, \ldots, x_n$, weights in $\mathbb{Q}^+$ for each input $w_1, \ldots, w_n$.

Solution: assignment α such that $C(\alpha(x_1), \ldots, \alpha(x_n)) = 1$.

Measure: $\max(1, \Sigma_{i=1}^{n} \alpha(x_i) w_i)$.

Type: maximization.

Theorem 3.3.3. *Maximum Variable-Weighted Satisfiability is complete for* NNCO *under logarithmic space AP reductions.*

Proof. This problem is complete for the class of maximization problems in NPO under polynomial-time AP reductions [55, Theorem 3.1]. A close inspection reveals that the functions of the reduction can be computed in logarithmic space. There is furthermore a polynomial-time AP reduction from the Minimum Variable-Weighted Satisfiability problem, which is complete for the class of all minimization problems in NPO, to Maximum Variable-Weighted Satisfiability [7, Theorem 8.4], and a close inspection of the reduction reveals that it can also be implemented in logarithmic space. Thus this problem is complete for NPO under logarithmic space AP reductions.

Next, we show that Maximum Variable-Weighted Satisfiability is in NNCO. The measure function is computable in FNC because the basic arithmetic operations and summation are both computable in FNC. The solution set is decidable in NC

because Boolean formula evaluation is computable in NC [12]. Since NNCO $\subseteq$ NPO we conclude that the problem is complete for NNCO. □

By converting a Boolean formula into its equivalent Boolean circuit, we get the following corollary.

Corollary 3.3.4. *Maximum Weighted Circuit Satisfiability is complete for* NPO *under logarithmic space AP reductions.*

An initial version of this theorem was suggested in [41].

Theorem 3.3.5. NNCO = NPO *if and only if* NC = P.

Proof. NNCO $\subseteq$ NPO by definition. If NC = P, then NNCO = NPO by definition. If NNCO = NPO, then Maximum Weighted Circuit Satisfiability is in NNCO, thus there is an NC algorithm that decides its solution relation. Its solution relation is precisely the Circuit Value problem, which is P-complete [40, Problem A.1.1]. An NC algorithm for a P-complete decision problem implies NC = P. □

Contrast this with the fact that NNC[poly] = NP [68, Theorem 2.2] (and in fact, $\mathsf{NNC}^1[\mathsf{poly}] = \mathsf{NP}$). So the classes of decision problems are equal whereas the classes of corresponding optimization problems are not, unless NC = P.

3.4 Completeness in classes of polynomial-time solvable problems

This section shows results nearly analogous to those in the previous section, but in the intersection of both NPO and NNCO with PO. The results here show completeness with respect to maximization problems only; we conjecture that both of the problems defined below are also complete with respect to minimization problems.

Definition 3.4.1 (Maximum Double Circuit Value)**.**

Instance: two Boolean circuits C_1 and C_2, binary string x.

Solution: binary string y such that $C_1(x) = y$ and $|x| + |y|$ equals the number of inputs to C_2.

Measure: $\max(1, C_2(x, y))$.

Type: maximization.

In this problem, the Boolean circuits may output binary strings of polynomial length interpreted as non-negative integers. This problem is constructed so that the circuit C_1 can simulate an algorithm that produces an optimal solution for an optimization problem and the circuit C_2 can simulate an algorithm that outputs the measure of a solution for that problem. Also, each input has exactly one solution, so this problem is quite artificial.

Definition 3.4.2 (Linear Programming)**.**

Instance: $m \times n$ integer matrix A, integer vector $\mathbf{b}$ of length m, integer vector $\mathbf{c}$ of length n.

Solution: non-negative rational vector $\mathbf{x}$ of length n such that $A\mathbf{x} \leq \mathbf{b}$.

Measure: $\max(1, \mathbf{c}^\intercal \mathbf{x})$.

Type: maximization.

Theorem 3.4.3. *Maximum Double Circuit Value is complete for the class of maximization problems in* PO *under logarithmic space AP reductions.*

Proof. Since Circuit Value is in P, both the solution and the measure function are computable in polynomial time. Therefore Maximum Double Circuit Value is in PO. Our goal is now to exhibit an AP reduction from any language in PO to Maximum Double Circuit Value. For the sake of brevity, suppose Maximum Double Circuit Value is defined by $(I_C, S_C, m_C, \max)$.

Let P be a maximization problem in PO, where $P = (I_P, S_P, m_P, \max)$. Let x be an element of I_P. Suppose E is the deterministic polynomial-time Turing machine that produces optimal solutions for P. Define f by $f(x) = (C_E, C_m, x)$ for all $x \in I_P$, where C_E is the Boolean circuit of polynomial size that simulates the action of E on input x and C_m is the circuit that simulates m_P on inputs x and $E(x)$. These circuits exist and are computable from x in logarithmic space [52]. Define g by $g(x, y) = y$ for all strings x and all y in $S_C(f(x))$. Let $\alpha = 1$.

Now, for any $x \in I_P$ and any $y \in S_C(f(x))$, we have

$$m_P(x, g(x, y)) = m_P(x, y) = C_m(x, y) = m_C((C_E, C_m, x), y) = m_C(f(x), y).$$

Since these measures are equal for all instances x and solutions y, we have shown that (f, g, α) is a logarithmic space AP reduction from P to Maximum Double Circuit Value. □

Theorem 3.4.4. *Linear Programming is complete for* PO *under logarithmic space AP reductions.*

Proof. Linear Programming is in PO by the ellipsoid algorithm [47]. We reduce Maximum Double Circuit Value to Linear Programming. The reduction is essentially the same as the reduction from Circuit Value to Linear Programming given (implicitly) in the hint beneath [40, Problem A.4.1]. We repeat it here for the sake of completeness.

Define the instance transducer f as follows. Suppose (C_1, C_2, x) is an instance of Maximum Double Circuit Value, and let $x = x_1 \cdots x_n$. For each of the circuits C_1 and C_2, the transducer f adds the following inequalities to the linear program.

1. For each bit of x, represent a 1 bit at index i by $x_i = 1$ and a 0 bit by $x_i = 0$.

2. Represent a NOT gate, $g = \neg h$, by the equation $g = 1 - h$ and the inequality $0 \leq g \leq 1$.

3. Represent an AND gate, $g = h_1 \wedge h_2$, by the inequalities $g \leq h_1$, $g \leq h_2$, $h_1 + h_2 - 1 \leq g$, and $0 \leq g \leq 1$.

4. Represent an OR gate, $g = h_1 \vee h_2$, by the inequalities $h_1 \leq g$, $h_2 \leq g$, $g \leq h_1 + h_2$, and $0 \leq g \leq 1$.

Suppose $y_1, \ldots, y_s$ are the variables corresponding to the output gates of C_1, and suppose $\mu_t, \ldots, \mu_1$ are the variables corresponding to the output gates of C_2, numbered from least significant bit to most significant bit (that is, right-to-left). The components of the object function $\mathbf{c}$ are assigned to be 2^i where the component corresponds to the variable μ_i and 0 everywhere else. The function f is computable in logarithmic space because the transformation can proceed gatewise, requiring only a logarithmic number of bits to record the index of the current gate. Suppose $\mathbf{x}$ is a solution to $f((C_1, C_2, x))$, that is, an assignment to the variables described above that satisfies all the inequalities. Define the solution transducer g by $g((C_1, C_2, x), \mathbf{x}) = y$, where $y = y_1 \cdots y_s$. This is also computable in logarithmic space by finding the index, in binary, of the necessary gates $y_1, \ldots, y_s$. Let $\alpha = 1$.

By structural induction on the gates of the circuits we see that a gate has value 1 on input x if and only if the solution vector $\mathbf{x}$ has a value 1 in the corresponding component, and $\mathbf{x}$ must be a vector over $\{0, 1\}$. Since the linear program correctly

simulates the circuits, we see that

$$\begin{aligned} m_A((C_1, C_2, x), g((C_1, C_2, x), \mathbf{x})) &= m_A((C_1, C_2, x), y) \\ &= C_2(x, y) \\ &= \mu_t \cdots \mu_1 \\ &= \Sigma_{i=1}^{t} 2^i \mu_i \\ &= m_B(f((C_1, C_2, x)), \mathbf{x}), \end{aligned}$$

where m_A is the measure function for Maximum Double Circuit Value and m_B is the measure function for Linear Programming. Since these measures are equal, we have shown that (f, g, α) is a logarithmic space AP reduction from Maximum Double Circuit Value to Linear Programming. Since the former is complete for the class of maximization problems in PO, so is Linear Programming. □

The reduction in the proof of Theorem 3.4.4 is more evidence that approximability is not closely related to the complexity of verification. Although Maximum Double Circuit Value is not in $\mathsf{PO} \cap \mathsf{NNCO}$ unless $\mathsf{NC} = \mathsf{P}$ (because its solution relation is P-complete), Linear Programming is not only in PO but also in NNCO, since matrix multiplication is in NC. This yields the following corollaries.

Corollary 3.4.5. $\mathsf{PO} \cap \mathsf{NNCO}$ *is not closed under logarithmic space AP reductions unless* $\mathsf{NC} = \mathsf{P}$.

Corollary 3.4.6. *Linear Programming is complete for the class of maximization problems in* $\mathsf{PO} \cap \mathsf{NNCO}$ *under logarithmic space AP reductions.*

An equivalence analogous to that of Theorem 3.3.5 also holds in the intersection with PO.

Theorem 3.4.7. $\mathsf{PO} \cap \mathsf{NNCO} = \mathsf{PO}$ *if and only if* $\mathsf{NC} = \mathsf{P}$.

Proof. $\mathsf{PO} \cap \mathsf{NNCO} \subseteq \mathsf{PO}$ by definition. If $\mathsf{NC} = \mathsf{P}$, then $\mathsf{PO} \cap \mathsf{NNCO} = \mathsf{PO}$ by definition. If $\mathsf{PO} \cap \mathsf{NNCO} = \mathsf{PO}$, then Maximum Double Circuit Value is in $\mathsf{PO} \cap \mathsf{NNCO}$, thus there is an NC algorithm that decides its solution relation. Its solution relation is a generalization of the Circuit Value problem, which is P-complete (as long as the length of the output y remains polynomial in the length of the input, this generalization remains P-complete). An NC algorithm for a P-complete decision problem implies $\mathsf{NC} = \mathsf{P}$. □

3.5 Hierarchies

The classes of approximable optimization problems are also likely distinct; this is well-known for polynomial-time approximability.

Theorem 3.5.1 ([7, Exercise 8.1])**.** *If* $\mathsf{P} \neq \mathsf{NP}$ *then*

$$\mathsf{PO} \subsetneq \mathsf{PTAS} \subsetneq \mathsf{ApxPO} \subsetneq \mathsf{NPO}.$$

A natural analog holds for NC approximation classes.

Theorem 3.5.2. *If* $\mathsf{NC} \neq \mathsf{NP}$ *then* $\mathsf{NCO} \subsetneq \mathsf{NCAS} \subsetneq \mathsf{ApxNCO} \subsetneq \mathsf{NNCO}$.

Proof. We begin by showing that $\mathsf{ApxNCO} = \mathsf{NNCO}$ implies $\mathsf{NC} = \mathsf{NNC}[\mathsf{poly}]$, and hence $\mathsf{NC} = \mathsf{NP}$. Let L be a decision problem complete for $\mathsf{NNC}[\mathsf{poly}]$ under logarithmic space many-one reductions (for example, Satisfiability). Suppose S_L is the relation decidable in NC and p is the polynomial such that $x \in L$ if and only if there is a string y of length $p(|x|)$ such that $(x, y) \in S_L$ for all strings x. Define the optimization

problem P by $P = (I, S, m, t)$, where

$$
\begin{aligned}
I &= \Sigma^*, \\
S &= \{(x, y) \mid |y| \leq p(|x|)\}, \\
m(x, y) &= \begin{cases} 1 & \text{if } (x, y) \in S_L, \\ 0 & \text{otherwise, and} \end{cases} \\
t &= \max .
\end{aligned}
$$

(Technically, the measure function must be positive; we can overcome this by translating the measure function up by some positive value.) Since S_L is in NC, the measure function m is in FNC. The sets I and S are trivially in NC, and S is polynomially bounded, so P is in $\mathsf{NNC}[\mathsf{poly}]$. By hypothesis P is also in ApxNCO, so there is an NC computable function A that is an r-approximator for P, for some constant $r \geq 1$. Assume without loss of generality that A enters a special state, call it $\perp$, if x has no solution in S_L.

Suppose x is a string that has a solution in S_L. Then $m^*(x) = 1$ and thus $m(x, A(x)) \geq \frac{1}{r} > 0$. Define a new algorithm D that, on input x, accepts if and only if $m(x, A(x)) > 0$. If x has a solution, then $m(x, A(x)) > 0$, otherwise A will output $\perp$ and D will reject. Furthermore, D is computable by an NC circuit because both A and m are. Therefore D is an NC circuit that decides L, so $\mathsf{NC} = \mathsf{NNC}[\mathsf{poly}]$.

If $\mathsf{NCAS} = \mathsf{ApxNCO}$ we can use a similar argument with $m(x, y) = \frac{1}{r}$ in the second case to produce $\mathsf{NC} = \mathsf{NP}$. This technique does not seem to work when attempting to prove $\mathsf{NCO} = \mathsf{NCAS}$ implies $\mathsf{NC} = \mathsf{NP}$. Instead we consider Maximum Independent Set for Planar Graphs; this problem is in NCAS [27, Theorem 5.2.1], and its budget problem is NP-complete [38]. Therefore an exact NC algorithm for it implies

$\mathsf{NC} = \mathsf{NP}$. □

As a corollary to this theorem, since Maximum Variable-Weighted Satisfiability is complete for the class NNCO under logarithmic space AP reductions, it admits no NC approximation algorithm unless $\mathsf{NC} = \mathsf{NP}$.

Theorem 3.5.3. *If* $\mathsf{NC} \neq \mathsf{P}$ *then*

$$\mathsf{NCO} \subsetneq \mathsf{PO} \cap \mathsf{NCAS} \subsetneq \mathsf{PO} \cap \mathsf{ApxNCO} \subsetneq \mathsf{PO} \cap \mathsf{NNCO} \subsetneq \mathsf{PO}.$$

Proof. From Theorem 3.4.7, we know that $\mathsf{PO} \cap \mathsf{NNCO} = \mathsf{PO}$ implies $\mathsf{NC} = \mathsf{P}$. If either $\mathsf{PO} \cap \mathsf{NCAS} = \mathsf{PO} \cap \mathsf{ApxNCO}$ or $\mathsf{PO} \cap \mathsf{ApxNCO} = \mathsf{PO} \cap \mathsf{NNCO}$, we can use the same technique as in Theorem 3.5.2. Instead of a problem complete for $\mathsf{NNC}[\mathsf{poly}]$, use a problem complete for P (which is a subset of $\mathsf{NNC}[\mathsf{poly}]$ anyway). Then the optimization problem P is in PO, but an NC approximation algorithm for it implies an NC algorithm for the decision problem L, and therefore $\mathsf{NC} = \mathsf{P}$.

Suppose now that $\mathsf{NCO} = \mathsf{PO} \cap \mathsf{NCAS}$. Consider Positive Linear Programming, the restriction of Linear Programming to only non-negative inputs. This problem is in PO (because it is a restriction of Linear Programming) and in NCAS [53]. However, its budget problem remains P-complete [67]. If $\mathsf{NCO} = \mathsf{PO} \cap \mathsf{NCAS}$, then there is an NC algorithm that solves this P-complete problem exactly, and hence $\mathsf{NC} = \mathsf{P}$. □

A similar proof shows that, for example, $\mathsf{PO} \cap \mathsf{ApxNCO} = \mathsf{ApxNCO}$ if and only if $\mathsf{P} = \mathsf{NP}$. It can be extended to $\mathsf{PO} \cap \mathsf{NCAS}$ as well. Compare this theorem with Theorem 3.4.7.

Theorem 3.5.4. $\mathsf{PO} \cap \mathsf{ApxNCO} = \mathsf{ApxNCO}$ *if and only if* $\mathsf{P} = \mathsf{NP}$.

Proof. The equation $\mathsf{PO} \cap \mathsf{ApxNCO} = \mathsf{ApxNCO}$ is equivalent to the inclusion $\mathsf{ApxNCO} \subseteq \mathsf{PO}$. For the reverse implication, $\mathsf{P} = \mathsf{NP}$ implies $\mathsf{PO} = \mathsf{NPO}$ by definition, and therefore $\mathsf{ApxNCO} \subseteq \mathsf{NNCO} \subseteq \mathsf{NPO} = \mathsf{PO}$. For the forward implication, the problem Maximum k-CNF Satisfiability problem is in ApxNCO by [7, Theorem 8.6]. By hypothesis, it is now in PO as well. Since its budget problem is NP-complete, we conclude that $\mathsf{P} = \mathsf{NP}$. □

Since Linear Programming is complete for the class of maximization problems in $\mathsf{PO} \cap \mathsf{NNCO}$ under logarithmic space AP reductions, it admits no NC approximation algorithm unless $\mathsf{NC} = \mathsf{P}$. This result suggests an explanation for the fact that r-approximating Linear Programming for any $r \geq 1$ is P-complete [27, Theorem 8.2.7], and further, the fact that any NC approximation algorithm for Linear Programming implies $\mathsf{NC} = \mathsf{P}$ [27, Theorem 8.2.8]: Corollary 3.4.6, Theorem 3.5.3, and the fact that AP reductions compose imply that any highly parallel approximation for Linear Programming necessitates $\mathsf{NC} = \mathsf{P}$.

The hierarchy theorem also provides a simple proof of a result of [27] (although they do not define ApxNCO in the same way).

Corollary 3.5.5 ([27, Theorem 8.2.9])**.** $\mathsf{ApxNCO} = \mathsf{ApxPO}$ *if and only if* $\mathsf{NC} = \mathsf{P}$.

Proof. If $\mathsf{NC} = \mathsf{P}$ then $\mathsf{ApxNCO} = \mathsf{ApxPO}$ by definition. If $\mathsf{ApxPO} \subseteq \mathsf{ApxNCO}$ then $\mathsf{PO} \subseteq \mathsf{NNCO}$, and hence $\mathsf{PO} \cap \mathsf{NNCO} = \mathsf{PO}$. By Theorem 3.5.3, we conclude that $\mathsf{NC} = \mathsf{P}$. □

This result is true if we replace the first equality with $\mathsf{NCAS} = \mathsf{PTAS}$, or, indeed, any equality that implies $\mathsf{PO} \subseteq \mathsf{NNCO}$.

3.6 Completeness in classes of approximable problems

In order to construct an optimization problem complete for, say, $\mathsf{PO} \cap \mathsf{ApxNCO}$, we need to use either

1. an analog of the PCP theorem with NC verifiers for polynomial-time decision problems, or

2. a canonical, "universal" complete problem for $\mathsf{PO} \cap \mathsf{ApxNCO}$.

These are the only two known ways for showing completeness in constant-factor approximation classes. The first approach is difficult to apply because it is not obvious how to construct a PCP for a deterministic time complexity class (PO). See chapter 2 for more information on that approach. The second approach is difficult to apply because although this technique has worked in the past for constructing a complete problem for ApxPO [23, Lemma 2], it is not clear how to guarantee a polynomial-time computable function that produces optimal solutions for such a problem.

However, we know what a problem complete for $\mathsf{PO} \cap \mathsf{ApxNCO}$ should look like. It should be exactly solvable in polynomial time and admit an NC approximation algorithm. It should also have threshold behavior in the following sense. If the problem were approximable for all $r > 1$, then it would be in NCAS. If the problem were not approximable for any $r \geq 1$, then it would not even be in ApxNCO. Therefore there should be some constant r_0 such that the problem is approximable for all $r \in (r_0, \infty)$ and not approximable for all $r \in (1, r_0)$.

3.6.1 High weight subgraph problems

There is in fact a family of maximization problems that has these properties: Induced Subgraph of High Weight for Linear Extremal Properties [27, Chapter 3].

A concrete example of a maximization problem in this family is Maximum High Degree Subgraph. In the definition below, the degree of a graph G, denoted $\deg(G)$, is defined by $\deg(G) = \min_{v \in V(G)} \deg(v)$.

Definition 3.6.1 (Maximum High Degree Subgraph)**.**

Instance: undirected graph G.

Solution: vertex-induced subgraph H.

Measure: $\deg(H)$.

Type: maximization.

This problem is exactly solvable in polynomial time, has an r-approximator in FNC for all $r \in (2, \infty)$ and has no r-approximator in FNC for all $r \in (1, 2)$ unless NC = P [2]. (The existence or non-existence of a 2-approximator seems to remain unknown.) We suspect this family of problems is complete for PO ∩ ApxNCO under logarithmic space AP reductions.

Conjecture 3.6.2. *Maximum High Degree Subgraph is complete for the class of maximization problems in* PO ∩ ApxNCO *under logarithmic space AP reductions.*

3.6.2 Restrictions of linear programming

Although Linear Programming is P-complete and admits no NC approximation algorithm, Positive Linear Programming, the restriction of Linear Programming to inputs in which all entries of A, $\mathbf{b}$, and $\mathbf{c}$ are non-negative, admits a NC approximation scheme [53], even though the corresponding budget problem remains P-complete [67, Theorem 4]. These results beg the question "is there some restriction of Linear Programming less strict than Positive Linear Programming that exhibits the properties of a complete problem for PO ∩ ApxNCO as defined above?" If we relax the non-negativity requirement and allow a small number of equality

constraints that can be violated by a small amount, then there is still an NC approximation scheme [67, Theorem 5.2]. On the other hand, if we have even just one equality constraint and don't allow the equality violations, the problem becomes hard to approximate again (to within a constant factor) [31, Theorem 3.1] [67, Remark 2]. Similarly, if we allow A to have negative entries, the problem is hard to approximate [30, Corollary 2]. The (γ, κ) form of Linear Programming is hard to approximate [30, Proposition 1]; the k-normal form reduces to Positive Linear Programming and so has an approximation scheme [65, Theorem 2].

There remains one candidate restriction that may have the properties we seek, the Linear Programming with Triplets problem. An instance of this maximization problem comprises $m \times n$ Boolean matrices $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$, non-negative rational vectors $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$ of length m, a non-negative rational vector $\mathbf{c}$ of length n, and a set of triples of indices $T \subseteq \{1, \ldots, n\}^3$ satisfying two conditions. First, there is at least one non-zero entry in each row of $A^{(1)}$ and each row of $A^{(2)}$. Second, there is a constant $\gamma \in (1, \infty)$ such that $M^* \leq \gamma M$ for all measures M of this instance, where M^* is the optimal measure of the instance. A valid solution is a pair of non-negative rational vectors $\mathbf{x}$ and $\mathbf{f}$ of length n such that for all $(i, j, k) \in T$,

$$\begin{aligned} x_k + (1 - f_i) &\leq 1 \\ x_k + (1 - f_j) &\leq 1 \\ f_k + f_i + f_j &\leq 2, \end{aligned}$$

as well as

$$A^{(1)}\mathbf{x} = \mathbf{b}^{(1)}$$

$$A^{(2)}\mathbf{x} + A^{(3)}\mathbf{f} = \mathbf{b}^{(2)}$$

$$\mathbf{x} \leq \mathbf{1}$$

$$\mathbf{f} \leq \mathbf{1}$$

The measure is $\max(1, \mathbf{c}^\intercal \mathbf{f})$.

Although this optimization problem is not approximable within $\frac{2n^\epsilon}{n^\epsilon+1}$ for any positive ϵ unless $\mathsf{NC} = \mathsf{P}$ [60, Corollary 1], we know the value of the optimal measure of an instance to within a multiplicative factor of γ.

Conjecture 3.6.3. *Linear Programming with Triplets is complete for the class of maximization problems in* $\mathsf{PO} \cap \mathsf{ApxNCO}$ *under logarithmic space AP reductions.*

3.6.3 Linear program for high degree subgraph

Is there a way to relax the high degree subgraph problem to make it easier to work with? Perhaps we can consider a linear programming relaxation of Maximum High Degree Subgraph and show that such a restriction is more approximable than Linear Programming but less approximable than Positive Linear Programming.

Suppose the vertices of a graph G are identified with the integers $\{1, \ldots, n\}$. We can represent a subgraph H of a graph as a subset of the vertices, and if the graph has n vertices, this can be an indicator vector $\mathbf{x}$ of length n. Since we also want to maximize the minimum degree of the chosen subgraph, we introduce a new variable d that has a value between 0 and n that will be bounded above by the degree of each vertex. We want the constraints to reflect that if a vertex is in the subgraph,

then the degree of that vertex (with respect to the subgraph) is at least d, and if a vertex is not in the subgraph then we don't care about its degree. In other words, we want "if $x_i = 1$ then $\deg(i) \geq d$," where $\deg(i) = \Sigma_{j=1}^{n} a_{ij}x_j$ and a_{ij} is the entry at row i, column j in the adjacency matrix of the graph. Equivalently, we want "$x_i \neq 1$ or $\deg(i) \geq d$," or more specifically, "$x_i < 1$ or $\deg(i) \geq d$." We can combine the two inequalities to get a single constraint "$x_i + d \leq 1 + \deg(x_i)$." However, as stated above we want this constraint to be always satisfied if $x_i = 0$ (that is, when the vertex i is not in the subgraph H); in this form, that is not always true.

We can assume without loss of generality that d, the minimum degree of H, will always be less than or equal to $n - 1$ (since a graph without self-loops cannot have a vertex of degree n anyway). Thus we can ensure the constraint is always satisfied if we modify it so that $x_i = 0$ implies d is less than the right side, "$nx_i + d \leq n + \deg(x_i)$." Now if $x_i = 1$ then $d \leq \deg(x_i)$ as required, and if $x_i = 0$ then $d \leq n - 1 \leq n \leq n + \deg(x_i)$ is always satisfied.

There is one final restriction: we require a non-empty subgraph, so we want at least one of the entries of $\mathbf{x}$ to be 1. Therefore, the proposed linear program is

$$\begin{aligned} &\text{maximize } d \\ &\text{subject to } n\mathbf{x} + d\mathbf{1} \leq n\mathbf{1} + A\mathbf{x} \\ &\mathbf{1}^{\intercal}\mathbf{x} > 0 \\ &\mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \\ &0 \leq d \leq n - 1, \end{aligned}$$

where A is the adjacency matrix of the graph G, n is the number of vertices in G, $\mathbf{0}$ is the all zeros vector, $\mathbf{1}$ is the all ones vector, and $\leq$ for vectors denotes component-wise

inequality. (If we restrict $\mathbf{x}$ and d to be integer-valued, then this is exactly the same problem.)

Thus the instances of the HDS Linear Programming maximization problem are an $m \times n$ Boolean matrix A, nonnegative rational vector $\mathbf{c}$ of length n, and a nonnegative rational c_0. The solution is a nonnegative rational vector $\mathbf{x}$ of length n and rational d satisfying the linear inequalities above. The measure is $\max(1, \mathbf{c}^\intercal \mathbf{x} + c_0 d)$.

Conjecture 3.6.4. *HDS Linear Programming is complete for* $\mathsf{PO} \cap \mathsf{ApxNCO}$ *under logarithmic space AP reductions.*

3.7 Syntactic characterization of ApxNCO

In [56], the authors introduce a wealth of problems which are complete under "L reductions" for MaxSNP, the class of maximization problems in strict NP (SNP), which is a syntactic characterization of a subclass of NP. Further work showed that the closure of MaxSNP under $\leq_E^P$ reductions, denoted $\mathrm{cl}\left(\mathsf{MaxSNP}, \leq_E^P\right)$, equals the subclass of ApxPO with polynomially-bounded measures, denoted ApxPO_{pb} [48, Theorem 1]. Since $\mathrm{cl}\left(\mathsf{ApxPO}_{pb}, \leq_{PTAS}^P\right) = \mathsf{ApxPO}$ [22], we conclude that $\mathrm{cl}\left(\mathsf{MaxSNP}, \leq_{PTAS}^P\right) = \mathsf{ApxPO}$ [48]. We also have $\mathrm{cl}\left(\mathsf{MaxSNP}, \leq_E^P\right) = \mathrm{cl}\left(\mathsf{MaxNP}, \leq_E^P\right)$ [48, Theorem 2] and Maximum Satisfiability is complete for MaxNP under $\leq_E^P$ reductions.

We conjecture that using descriptive complexity theory, these findings can translate to NC. For example, we know that $\mathsf{FO[polylog]} = \mathsf{NC}$ [44, Theorem 5.2], which may be used to construct a syntactic definition of ApxNCO. Such a definition may help to construct a complete problem for ApxNCO using a different strategy than that of the previous section.

Chapter 4

Parameterized problems

According to researchers Downey and Fellows in the introduction to [29], "The future of algorithms is multivariate." Their suggestion is to replace the classical viewpoint of computational problems as univariate objects with a more modern viewpoint of computational problems as multivariate objects. Understanding the parameterization for a computational problem allows us to break algorithms for the problem into parts and more easily identify the complexity of these parts.

In the world of parallel versus sequential computation, the right parameterization of a problem that would classically be considered inherently sequential can yield a highly parallel algorithm. Little work has been done to provide the framework for proving parameterized parallelizability for classical inherently sequential computational problems; previous work has mostly focused on parameterized tractability for classically intractable computational problems. Practicioners should be able to take advantage of parallelism where it exists, and this line of research may reveal a hidden capacity for parallelism that was previously unknown.

With this need in mind, we undertake the first comprehensive examination of the

definitions, supporting lemmas, and basic structural theorems about highly parallel parameterized problems and inherently sequential parameterized problems; this chapter provides such a framework.

Our work builds on the parameterized complexity theory framework that was originally intended to study the difference between tractable and intractable parameterized problems. Among other things, we prove

- the existence of inherently sequential problems whose parameterized versions are parallelizable (subsection 4.3.2),
- the existence of inherently sequential problems whose parameterized versions are not parallelizable under a reasonable assumption (subsection 4.4.1),
- the existence of a hierarchy of complete problems interpolating between the parameterized versions of the formula satisfiability problem and the circuit satisfiability problem (subsection 4.5.3),
- an equivalence between parameterized parallel verifiability and classical limited nondeterminism (subsection 4.5.4).

Altogether, these results demonstrates a strong relationship among the resources time, nondeterminism, and parallelism. We hope this framework inspires researchers to look more closely for paralellizable problems by considering parameterizations of classical problems previously considered inherently sequential.

4.1 History

The study of parameterized complexity as a distinct named concept was initiated in a series of articles by Downey and Fellows in the early 1990s; references can be found in their 2013 book *Fundamentals of Parameterized Complexity* [29] or in the 2006 book

by Flum and Grohe *Parameterized Complexity Theory* [35]. Much of our definitions, notation, and concepts follow the view outlined in the latter book. The first formal definition for a notion of fixed-parameter parallelizability seems to be in a 1998 article by Cesati and Di Ianni [18]. That work inspired our complete problems for paraP in subsection 4.4.1. Although much research has been done with respect to the class paraP, we could not find any proof of the existence of a complete problem.

A generic para operator that can be applied to any complexity class was defined by Flum and Grohe in 2003 [34]. In 2014, Elberfeld, Stockhusen, and Tantau explored this generic operator in more detail [32], giving complete problems for a number of parameterized complexity classes, including $\mathsf{paraWNC}^1$ (see Definition 4.5.1 for the definition of this class). We follow their lead when proving complete problems for $\mathsf{paraWNC}^k$ in subsection 4.5.3 below. More recently, a 2015 preprint by Bannach, Stockhusen, and Tantau [8] studies the parameterized depth of a circuit in more detail, showing that depth $f(k)$ is strictly more powerful than depth $O(1)$.

The deterministic simulations of nondeterminism in the parameterized and decision complexity classes appearing in Theorem 4.5.15 is inspired by similar theorems for polynomial-time computations by Cai et al. in 1995 [16] and Cai and Chen in 1997 [14], from which we have refactored the main components of the proofs into distinct lemmas.

4.2 Definitions

This section provides a brief introduction to the concepts necessary for studying parameterized complexity theory specifically for classes of problems decidable by families of Boolean circuits. For a more thorough review of the basics of computational complexity theory, see for example [3]; for parameterized complexity theory, see [29]

or [35].

Definition 4.2.1. A *Boolean circuit*, or simply a *circuit*, C, is a directed acyclic graph. The *size of a circuit*, denoted size(C), is the number of vertices in the underlying graph. The *depth of a circuit*, denoted depth(C), is the length of a longest path from the root to a sink.

Definition 4.2.2. A function f is *circuit-computable* if there is a nonuniform family of Boolean circuits $\{C_n\}_{n\in\mathbb{N}}$ such that for each x we have $f(x) = C_n(x)$, where $n = |x|$.

A language is *circuit-decidable* if it has a circuit-computable characteristic function. We may also require that the size and depth of each circuit C_n in the family be circuit-computable from just n, the length of the input. In this case, we say the language is *circuit-decidable with uniform size and depth*.

Nonuniformity is required in Lemma 4.3.7, among other theorems, in which the size of the input relative to the size of the parameter for an instance of the parameterized problem selects which of two circuits to use; for more information, see the footnote in the referenced lemma. However, it seems that all other theorems can be made uniform, with reasonable restrictions on the complexity of the parameterization. It may be possible to adapt some theorems in later sections to use uniform circuits, but we did not pursue this.

Definition 4.2.3 (Decision problems and parameterized problems)**.** A *language* is a set of binary strings. A *parameterization* is a computable function κ from binary strings to natural numbers. A *parameterized problem* is a pair (Q, κ), where Q is a language and κ is a parameterization.

Definition 4.2.4 (Slices of parameterized problems)**.** For each positive integer k and

each parameterized problem (Q, κ), the *kth slice of* Q, denoted $(Q, \kappa)_k$, is defined by

$$(Q, \kappa)_k = \{(x, k) \mid x \in Q \text{ and } \kappa(x) = k\}.$$

Definition 4.2.5 (NNC^d and NC^d)**.** Let d be a natural number. A language Q is in the class $\mathsf{NNC}^d[b(n)]$ if there is a nondeterministic circuit family $\{C_n\}$ such that for each string x of length n,

- $x \in Q$ if and only if $C_n(x) = 1$,
- $\text{size}(C_n) \leq n^{O(1)}$,
- $\text{depth}(C_n) \leq O(\log^d n)$,
- $\text{nondet}(C_n) \leq b(n)$.

If b is the zero function, then the language is in the class NC^d.

Here, the notion of "acceptance" for a circuit is a nondeterministic one: $C_n(x) = 1$ if and only if there is a binary string w of length $b(n)$ such that $C_n(x, w) = 1$.

Throughout we will often assume without loss of generality that functions like circuit size and depth bounds, nondeterministism bounds, and polynomials, are increasing.

4.3 Fixed-parameter parallelizability

Classical computational complexity has a well-developed theory of parallel versus sequential computation. Those computational problems that are P-complete are inherently sequential, whereas those in the class NC admit highly parallel algorithms. Even though they can be solved in polynomial time, adding more processors does not provide any significant reduction in the time required to find a solution for P-complete

problems. Can parameterization of problems that have traditionally been considered inherently sequential afford us a new avenue for parallelization?

Specifically, we would like to determine whether there are problems that are inherently sequential in the classical sense but parallelizable under some parameterization. This section combines and adapts the definitions of parallelization from classical complexity theory and the definitions of parameterized complexity theory for highly parallel problems.

We provide the definition of paraNC, demonstrate the relationship between paraNC and NC, and prove some sufficient conditions for membership in paraNC. These results demonstrate that the idea of parameterized parallelizability is both meaningful and interesting for seemingly inherently sequential problems. Subsequent sections will examine the limits to parameterized parallel computation.

4.3.1 Definition of paraNC

The para "operator" defined in [34] applies generically to an arbitrary complexity class as follows. If $\mathcal{C}$ is a class of decision problems, then para$\mathcal{C}$ is the class of parameterized problems (Q, κ) for which there is a decision problem $L \in \mathcal{C}$ and a computable function f such that x in Q if and only if $(x, 1^{f(\kappa(x))}) \in L$. When $\mathcal{C} = \mathsf{NC}$ in particular, we get the following equivalent definition.

Definition 4.3.1 (paraNC^d)**.** Let d be a natural number. A parameterized problem (Q, κ) is in the class paraNC^d if there is a circuit-computable function f and a nonuniform family $\{C_{n,k}\}$ of bounded fan-in Boolean circuits such that for each string x,

- $x \in Q$ if and only if $C_{n,k}(x) = 1$, where $n = |x|$ and $k = \kappa(x)$,
- size$(C_{n,k}) \leq f(k)n^{O(1)}$,

- $\text{depth}(C_{n,k}) \leq f(k) + O(\log^d n)$.

If the depth of the circuit is instead bounded by $f(k)O(\log^d n)$, the class is denoted $\mathsf{paraNC}^{d\uparrow}$, a superclass of paraNC^d. If the circuits are of unbounded fan-in, the classes are paraAC^d and $\mathsf{paraAC}^{d\uparrow}$, respectively. The class $\mathsf{paraAC}^{d\uparrow}$ was first defined in [8].

A subtle point is that the value of the parameter $\kappa(x)$ must be non-constant but also independent of the size of the instance x for the parameterized problem to be interesting. First, if $\kappa(x)$ were bounded above by a constant for each x, then the parameter would be irrelevant and the problem would simply be in the standard complexity class NC^d. Thus depth $O(\log^d n)$ and depth $f(k) + \log^d n$ (as well as $f(k)\log^d n$) are different. On the other hand, if $\kappa(x)$ were bounded from below by a nondecreasing, unbounded function of $|x|$, then the problem would be trivially in $\mathsf{paraAC}^{0\uparrow}$ by the technique of [35, Proposition 1.7]. Thus a formula like $\log^2(kn)$, which may appear in the analysis of certain simulations of parameterized complexity classes (see Lemma 4.5.10, for example), becomes

$$\log^2(kn) = (\log k + \log n)^2 = \log^2 k + \log k \log n + \log^2 n \leq \log^2 k + 2\log^2 n,$$

and thus $\log^2(kn) = f(k) + O(\log^2 n)$ for some computable function f.

4.3.2 Example problem in paraNC

Useful complexity classes are nonempty and have interesting natural problems, so in an effort to show that paraNC has parameterized problems whose underlying decision problems are P-complete, we consider parameterizations of the canonical P-complete problem, the circuit evaluation problem. This section provides a non-degenerate parameterization of the circuit evaluation problem that makes it parallelizable.

We provide a paraNC problem based on a P-complete problem with a degenerate

parameterization (Theorem 4.3.2), one with a nondegenerate parameterization (Theorem 4.3.3), and one based on an optimization problem (Theorem 4.3.4). (Contrast these results with the parameterized vertex cover problem, which is in paraAC^0 [8], but whose underlying decision problem is NP-complete.) Thus paraNC does indeed contain interesting problems and we can use these as the starting point for studying the limits of parameterized parallelization.

We start by choosing Q to be a P-complete problem and κ to be the "degenerate" parameterization function $\kappa(x) = |x|$. The *circuit evaluation problem* is the problem of deciding whether, given a Boolean circuit and an input to that circuit, the output of the circuit is 1.

Theorem 4.3.2. *The circuit evaluation problem parameterized by the size of the circuit is in* paraNC *and the underlying decision problem is* P*-complete.*

Proof. The circuit evaluation problem is P-complete by [52]. Since the parameterization is monotonically increasing with the size of the input, the problem is in paraNC by the technique of [35, Proposition 1.7]. □

To find a non-degenerate example, we can parameterize the circuit evaluation problem by depth instead of size.

Theorem 4.3.3. *The circuit evaluation problem parameterized by the depth of the circuit is in* $\mathsf{paraAC}^{0\uparrow}$ *and the underlying decision problem is* P*-complete.*

Proof. As stated in the proof of the previous theorem, the circuit evaluation problem is P-complete. Evaluating the circuit C of size m and depth d on inputs x can be performed by the depth-universal circuit U of [20]. The size of U is $O(m)$ and the depth is d, so there is a function f such that the size is bounded by $f(d)m^{O(1)}$ and the depth by $f(d)$. Therefore the circuit evaluation problem parameterized by circuit depth is in $\mathsf{paraAC}^{0\uparrow}$. □

For a problem with a standard parameterization derived from an optimization problem (see Definition 4.3.12 below), consider the "depth of ones" problem. The *depth of ones* problem is the problem of deciding, given a circuit, an input to the circuit, and a positive integer k, whether a 1 appears at depth at least k when evaluating the circuit on the input. The circuit evaluation problem is a special case of the depth of ones problem if we choose k to be the depth of the circuit C. As an optimization problem, the depth of ones problem is inapproximable up to any constant factor by any NC circuit, unless NC = P [50]. Contrast this with the complexity of the corresponding parameterized problem.

Theorem 4.3.4. *The depth of ones problem parameterized by the depth parameter k is in* paraAC$^{0\uparrow}$ *and the underlying decision problem is* P*-complete.*

Proof. Computing the depth of ones in a circuit is P-complete [50] (see also [40, Problem A.1.10]). The naïve algorithm for solving this problem is to take the subcircuit consisting of all gates starting from the inputs and extending through layer k, evaluating that (multi-output) circuit, then applying a single OR gate to decide whether any of the gates at layer k evaluated to one. For each gate at layer k, use an instance of the depth-universal circuit to evaluate the single-output circuit induced by that gate. This yields a circuit of depth $O(k)$ and size $f(k)m^{O(1)}$ for some f, where m is the size of the circuit given as input. Therefore this problem is in paraAC$^{0\uparrow}$. □

4.3.3 Relationship between paraNC and NC

How do the parallelizable parameterized problems relate to classical parallelizable computational problems? In order to determine the conditions under which a parameterized parallel algorithm implies a classical parallel algorithm (and vice versa), we consolidate and adapt some results that appear scattered across several parame-

terized complexity papers and books. This section provides a generic technique for constructing a classical parallel algorithm from a parameterized one, and vice versa.

Specifically, we show two main lemmas. Lemma 4.3.7 shows how to construct a parameterized parallel algorithm from a parameter-restricted reduction to a highly parallel decision problem. Lemma 4.3.9 shows how to construct a classical parallel algorithm for from a parameter-restricted reduction to a parameterized parallel problem. These give explicit techniques for transforming a parameterized parallel algorithm into a classical parallel algorith and vice versa. They will be used in later sections to provide evidence against the collapse of larger complexity classes to paraNC.

We begin with a lemma that allows us to construct a function that behaves like an upper bound on the inverse of another function.

Lemma 4.3.5. *For each nondecreasing, unbounded, circuit-computable function i, there is a function f_i such that $f_i(i(n)) \geq n$ for each $n \geq f(1)$. Furthermore, f_i is nondecreasing, unbounded, and circuit-computable. (We call f_i the "upper inverse" of i.)*

Proof. Define f_i by

$$f_i(k) = \max\{n_0 \in \mathbb{N} \mid \forall n \geq n_0 \colon i(n) \geq k\}.$$

Since i is nondecreasing and unbounded, so is f_i.

To compute f_i, we use the fact that i is nondecreasing is unbounded. We know that for each k there is a natural number n_k such that for all $n \geq n_k$, we have $i(n) \geq k$. Thus the algorithm for computing f_i takes k as input and performs a binary search on $i(1), \ldots, i(n_k)$ to determine the largest n such that $i(n) \geq k$. There will be at most $\log n_k$ comparison subcircuits, each requiring a computation of i and a comparison

with the integer k (in binary, say), so the overall depth of the circuit computing f_i is $O(\text{depth}(i) \log n \log\log k)$ and the size is $O(\text{size}(i) \log n \log k)$. □

Definition 4.3.6. Suppose d is a natural number, (Q, κ) is a parameterized problem, and Q' is a decision problem. There is a *small parameter* NC^d *many-one reduction* from (Q, κ) to Q' if there is a nondecreasing, unbounded, circuit-computable function i and an NC^d family of circuits $\{R_n\}_{n\in\mathbb{N}}$ such that for each string x of length n with $\kappa(x) \leq i(n)$, we have $x \in Q$ if and only if $R_n(x) \in Q'$.

This lemma essentially demonstrates that the closure of NC under small parameter reductions is a subset of paraNC. We attempted to show that the closure equals paraNC but were unable to do so.

Lemma 4.3.7. *Suppose d is a natural number, (Q, κ) is a parameterized problem, and Q' is a decision problem. If Q is circuit-decidable with uniform size and depth, Q' is in NC^d, and there is a small parameter NC^d many-one reduction from (Q, κ) to Q', then (Q, κ) is in paraNC^d.*

Proof. Let i be the function that defines the upper bound on the parameter, below which there is an NC^d many-one reduction from Q to Q'. Let $\{R_n\}$ be the NC^d circuit family computing the reduction. The nonuniform family of circuits $\{A_{n,k}\}$ that decides (Q, κ) is defined by

$$A_{n,k} = \begin{cases} C^1_n & \text{if } i(n) < k \\ C^2_{n'} \circ R_n & \text{otherwise,} \end{cases}$$

where $\{C^1_n\}$ is the family of circuits that decides Q with uniform size and depth, $\{C^2_n\}$ is the family of NC^d circuits that decides Q', and n' is the number of output bits of R_n. The correctness of $A_{n,k}$ follows from the correctness of the subsequent circuits. The circuit family is necessarily nonuniform: the computation of $i(n)$ and k and the

comparison of the two decides nonuniformly which circuit to select when defining $A_{n,k}$.

If $i(n) \geq k$, then the size and depth of the circuit are polynomial and polylogarithmic in n, respectively, because the size and depth of $C^2_{n'}$ and R_n are. For the case when $i(n) < k$, consider the upper inverse f_i of i guaranteed by Lemma 4.3.5. By construction, $n \leq f_i(i(n)) < f_i(k)$. Now

$$\begin{aligned} \text{size}(A_{n,k}) &= \text{size}(C^1_n) = S(n) \leq S(f_i(k)), \\ \text{depth}(A_{n,k}) &= \text{depth}(C^1_n) = D(n) \leq D(f_i(k)), \end{aligned}$$

where S and D are the (circuit-computable, nondecreasing) size and depth bounds for the circuit family $\{C^1_n\}$. Thus in either case, there is a sufficiently large circuit-computable function f such that the size of $A_{n,k}$ is bounded above by $f(k)n^{O(1)}$ and the depth $f(k) + O(\log^d n)$. □

As an aside, let us consider the nonuniformity requirement in this lemma. All subsequent theorems that require nonuniform circuits are nonuniform because they rely on this lemma, and it is not clear whether this lemma can be made uniform. Nonuniformity is necessary here, but only a *single bit* of nonuniform advice is required to select the appropriate circuit for $A_{n,k}$, given the length n and the parameterization k of the input. If the circuit $A_{n,k}$ were implemented with a selector for $i(n) < k$ and both branches as subcircuits, then the overall depth of the circuit would be bounded above by the larger of the depths of the two subcircuits. For large values of k, this could be too great a depth to qualify as paraNC.

On the other hand, if the function i were computable by, for example, a deterministic logarithmic space Turing machine, then we would be able to conclude that (Q, κ) is in paraL-uniform paraNC (assuming we have an appropriate definition for such a class). In this paper, i will not have that restriction, so we suggest considering parameterized

uniformity in future research.

We continue with a corollary of the previous lemma. The following corollary highlights the special case of the preceding lemma in which the reduction is the identity function.

Corollary 4.3.8. *Suppose (Q, κ) is a parameterized problem, d is a positive integer, and i is an unbounded, nondecreasing, circuit-computable function. Let $i(n)$-Q denote the problem of deciding, given x with $\kappa(x) \leq i(|x|)$, whether $x \in Q$. If $i(n)$-Q is in NC^d, then (Q, κ) is in paraNC^d.*

Proof. The identity function is a small parameter NC^d many-one reduction from (Q, κ) to $i(n)$-Q, thereby proving that Q is in paraNC^d by the previous lemma. □

This lemma shows that a many-one reduction to a fixed-parameter parallelizable problem can sometimes induce a highly parallel algorithm, if the parameter functions are bounded for the reduced instance.

Lemma 4.3.9. *Suppose d is a positive integer, Q is a decision problem, and (Q', κ') is a parameterized problem. Suppose there is an NC^d many-one reduction from Q to Q', given by the circuit family $\{R_n\}$, and (Q', κ') is in paraNC^d by a circuit family $\{C_{m,k}\}$ of size $f(k)m^{O(1)}$ and depth $f(k) + O(\log^d m)$ on inputs of length m. If $f(\kappa'(R_n(x))) \leq \min(n^{O(1)}, O(\log^d n))$, then Q is in NC^d.*

Proof. The circuit family that decides Q is $\{A_n\}$, defined by $A_n = C_{m,k} \circ R_n$, where m is the size of the output of R_n and $k = \kappa'(R_n(x))$. Since $\text{size}(R_n) = n^{O(1)}$, we have $m = n^{O(1)}$ as well. For correctness,

$$x \in Q \iff R_n(x) \in Q' \iff C_{m,k}(R_n(x)) = 1.$$

For size and depth bounds,

$$\begin{aligned}
\text{size}(A_n) &= \text{size}(C_{m,k}) + \text{size}(R_n) \\
&= f(k)m^{O(1)} + n^{O(1)} \\
&= f(k)n^{O(1)} + n^{O(1)} \\
&= n^{O(1)}n^{O(1)} + n^{O(1)} \\
&= n^{O(1)},
\end{aligned}$$

and

$$\begin{aligned}
\text{depth}(A_n) &= \text{depth}(C_{m,k}) + \text{depth}(R_n) \\
&= f(k) + O(\log^d m) + O(\log^d n) \\
&= f(k) + O(\log^d n) + O(\log^d n) \\
&= f(k) + O(\log^d n) \\
&= O(\log^d n) + O(\log^d n) \\
&= O(\log^d n).
\end{aligned}$$

□

The following corollary highlights the special case of the preceding lemma in which the decision problem of interest is a "bounded-parameter" version of the decision problem underlying the fixed-parameter parallelizable problem; compare this with Corollary 4.3.8. Below, a "nontrivial" parameterized problem is one in which $\emptyset \subsetneq Q \subsetneq \{0,1\}^*$.

Corollary 4.3.10. *Suppose (Q, κ) is a nontrivial parameterized problem, d is a positive integer, and i is an unbounded, nondecreasing, circuit-computable function. Let $i(n)$-Q denote the problem of deciding, given x with $\kappa(x) \leq i(|x|)$, whether $x \in Q$. If (Q, κ)*

is in paraNC^d *by a circuit family* $\{C_{n,k}\}$ *of size* $f(k)n^{O(1)}$ *and depth* $f(k) + O(\log^d n)$ *on inputs of length* n *and* $f(i(n)) \leq \min(n^{O(1)}, O(\log^d n))$*, then* $i(n)$*-*Q *is in* NC^d*.*

Proof. We will show a many-one reduction from the decision problem $i(n)$-Q to the decision problem Q underlying the parameterized problem (Q, κ) that satisfies the conditions of the previous lemma. The reduction $\{R_n\}$ is defined as follows.

$$R_n(x) = \begin{cases} x & \text{if } \kappa(x) \leq i(n), \\ \bot & \text{otherwise,} \end{cases}$$

where $\bot$ is an arbitrary string not in Q (which must exist because the problem is nontrivial by hypothesis). As long as κ is computable by an NC^d circuit family, then so is R_n. (The computation of $i(n)$ is captured by the nonuniformity of the circuit family, so it does not affect the size or depth required by the circuit computing R_n.)

The reduction R_n is a correct many-one reduction. If $x \in i(n)$-Q, then $\kappa(x) \leq i(|x|)$ and $x \in Q$, thus $R_n(x) \in Q$. If $x \notin i(n)$-Q, then there are two cases. In the first, $\kappa(x) > i(|x|)$, in which case $R_n(x) = \bot$, which is not in Q by construction. In the second case, $x \notin Q$ so $R_n(x) \notin Q$.

Finally, we consider the value of $f(\kappa(R_n(x)))$. If $\kappa(x) \leq i(n)$, then by construction

$$f(\kappa(R_n(x)) \leq f(\kappa(x)) \leq f(i(n)) \leq \min(n^{O(1)}, O(\log^d n)).$$

On the other hand, if $\kappa(x) > i(n)$, then $\kappa(R_n(x)) = \kappa(\bot) = O(1)$, which is bounded above by both $n^{O(1)}$ and $O(\log^d n)$ for all but finitely many n. Thus we have shown that $f(\kappa(R_n(x)))$ satisfies the upper bound required by Lemma 4.3.9 and the conclusion, $i(n)$-Q is in NC^d, follows. □

4.3.4 Approximable optimization problems

Theorem 4.3.4 shows an inherently sequential optimization problem that becomes parallelizable when parameterized. Let us explore the possibility of parameterized parallel algorithms from optimization problems more generally. (This has been done before only for efficient algorithm for intractable optimization problems.) We show how a certain kind of approximation scheme for an optimization problem induces a highly parallel algorithm for the standard parameterized problem derived from the optimization problem. This section provides the necessary definitions and generic theorems for this framework.

We prove that an approximation scheme with appropriate size and depth bounds implies a paraNC algorithm (Theorem 4.3.18), and show how this applies to the maximum flow problem under a derandomization assumption (Theorem 4.3.20). This means that both existing and newly discovered approximation schemes for optimization problems may present an alternate method of parallelization (via parameterization). One thing we are unable to show in this section is an optimization problem whose budget problem is P-complete and whose standard parameterization is in paraNC but for which no ENCAS exists, so we postpone that for future work.

We start with the necessary definitions for optimization problems and approximation schemes. Some of these definitions appear in chapter 3, but we repeat them here in a more concise form so that this section is self-contained.

Definition 4.3.11. An *optimization problem* O is a four-tuple (I, S, m, t), where I is the set of instances, S is the set of pairs (x, w) where w is a solution for x, the function m computes the *measure* (or *objective value*) for such a pair, and t is either min or max.

Definition 4.3.12. The *standard parameterization* of a minimization problem O,

denoted p-O, is (Q, κ), where $Q = \{(x, k) \mid m^*(x) \leq k\}$ and $\kappa(x, k) = k$. The inequality is reversed for a maximization problem.

Definition 4.3.13. Suppose (I, S, m, t) is an optimization problem and $(x, y) \in S$. The *performance ratio* of the solution y (with respect to x), denoted $R(x, y)$, is defined by

$$R(x, y) = \max\left(\frac{m(x, y)}{m^*(x)}, \frac{m^*(x)}{m(x, y)}\right)$$

The performance ratio $R(x, y)$ is a number in the interval $[1, \infty)$. The closer $R(x, y)$ is to 1, the better the solution y is for x, and the closer $R(x, y)$ to ∞, the worse the solution.

Definition 4.3.14. An *approximation scheme* for an optimization problem is a function A such that for all x and all positive integers k we have $(x, A(x, k)) \in S$ and $R(x, A(x, k)) \leq 1 + \frac{1}{k}$.

An approximation scheme induces a family of functions, $\{A_k\}_{k \in \mathbb{N}}$, that form progressively better approximations for the optimization problem. A problem is in NCAS if it admits an approximation scheme whose slices are in NC. The problem is in FNCAS if it admits an approximation scheme that is in NC with respect to both inputs n and k.

Definition 4.3.15. Suppose O is an optimization problem with $O = (I, S, m, t)$ with I and S in NC and m in FNC. An optimization problem O is in NCAS if there is an approximation scheme A for O such that for each k, we have $A_k \in$ FNC, where $A_k(x) = A(x, k)$ for each x. The problem is in FNCAS if there is an approximation scheme A for O such that $A \in$ FNC (i.e. on both inputs).

These two complexity classes lead us to a natural interpolation using the ideas of parameterized complexity theory. This definition is adapted from [35, Definition 1.31]

Definition 4.3.16 (ENCAS)**.** An optimization problem O is in ENCAS if there is a circuit family $\{A_{n,k}\}$ and a circuit-computable function f such that

- $\{A_{n,k}\}$ is an approximation scheme for O,
- $\text{size}(A_{n,k}) \leq f(k)n^{O(1)}$,
- $\text{depth}(A_{n,k}) \leq f(k) + \log^{O(1)} n$.

When we consider k as a parameter, then ENCAS interpolates between FNCAS and NCAS. If $f(k)$ is polylogarithmic in n, then the definition yields FNCAS. If $f(k)$ is considered a fixed constant, then the definition yields NCAS.

Proposition 4.3.17. FNCAS $\subseteq$ ENCAS $\subseteq$ NCAS.

We can use an ENCAS algorithm to construct a paraNC algorithm for the standard parameterization of an optimization problem. The converse does not hold: as a counterexample, the minimum vertex cover problem is not in ENCAS (since no polynomial-time approximation algorithm with approximation ratio better than 7⁄6 exists [42, Theorem 8.1]) but the problem is in paraNC [8, Theorem 4.5]. This theorem is an adaptation of [35, Theorem 1.32].

Theorem 4.3.18. *Let O be an optimization problem. If O is in* ENCAS*, then p-O is in* paraNC*.*

Proof. Assume without loss of generality that O is a minimization problem; the proof is similar if it is a maximization problem. Let $\{m_n\}$ be the NC circuit family that computes the measure function. Let $\{A_{n,k}\}$ be the circuit family such that

- $R(x, A_{n,k}(x,k)) \leq 1 + \frac{1}{k}$ for each x and k,
- $\text{size}(A_{n,k}) \leq f(k)n^{O(1)}$,

- $\text{depth}(A_{n,k}) \leq f(k) + O(\log^{O(1)} n)$,

for some circuit-computable function f. Define the circuit family $\{C_{n,k}\}$ as

$$C_{n,k}(x,k) = 1 \iff m(x, A_{n,k+1}(x, k+1)) \leq k,$$

so $C_{n,k}$ outputs 1 if and only if the approximate solution corresponding to parameter $k+1$ measures less than $k+1$. (The function m is really a circuit as well, chosen from a family of circuits depending on the number of bits in its inputs.)

The size of $C_{n,k}$ is $O(\text{size}(m) + \text{size}(A_{n,k+1}))$ and the depth is $O(\text{depth}(m) + \text{depth}(A_{n,k+1})$. For some sufficiently large circuit-computable function f', the size and depth bounds are $f'(k+1)n^{O(1)}$ and $f'(k+1) + O(\log^{O(1)} n)$, respectively. It remains to show correctness of $C_{n,k}$.

Let x be a string, let k be a natural number, and let $y = A_{n,k+1}(x, k+1)$. If $C_{n,k} = 1$, then $m(x,y) \leq k$, so $m^*(k) \leq k$ and therefore $(x,k) \in p\text{-}O$. For the converse, if $C_{n,k} = 0$, then $m(x,y) \geq k+1$, so

$$m^*(x) \geq \frac{m(x,y)}{1 + \frac{1}{k+1}} \geq \frac{k+1}{1 + \frac{1}{k+1}} = \frac{(k+1)^2}{k+2} > k.$$

Thus $(x,k) \notin p\text{-}O$. Therefore, we conclude that $p\text{-}O$ is in paraNC. □

Our goal now reduces to finding an optimization problem in ENCAS whose budget problem is P-complete. We can provide one under a derandomization assumption.

Definition 4.3.19 (Maximum Flow)**.**

Instance: directed graph G, a natural number capacity c_e for each edge e, source node s, and target node t.

Solution: flow F, defined as a real number F_e for each edge e such that $F_e \leq c_e$ and at each vertex the total in-flow is at least the total out-flow.

Measure: total in-flow at t.

Type: maximization.

Theorem 4.3.20. *If* NC = RNC*, then the budget problem for* Maximum Flow *is* P*-complete and the standard parameterization is in* paraNC*.*

Proof. The budget problem for Maximum Flow is P-complete [40, Problem A.4.4]. The Maximum Flow problem is in randomized FNCAS [27, Theorem 4.5.2]. If NC = RNC, then randomized FNCAS equals deterministic FNCAS. Thus, the problem is in ENCAS, by Proposition 4.3.17. Finally, the standard parameterization is in paraNC by Theorem 4.3.18. □

4.4 Fixed-parameter tractability

In classical complexity, there are limits to parallel computation; for some computational problems, adding more processors does not help solve the problem significantly more quickly. With the definition of parameterized parallel computation given in the previous section, we now must ask whether there are similar limitations for parameterized problems. We adapt our understanding of classical complexity to answer this question in the affirmative. This section demonstrates that under an appropriate parameterization, the natural inherently sequential classical problems become inherently sequential parameterized problems.

The definition of paraP (also known as FPT) is analogous to that of paraNC. Unlike

for paraNC, however, there are numerous examples of natural parameterized problems in paraP; see the listing in [17], for example.

Definition 4.4.1. A parameterized problem (Q, κ) is in paraP if there is a deterministic Turing machine M, a polynomial p, and a computable function f such that M decides Q within $f(\kappa(x))p(n)$ steps.

We prove the existence of paraP-complete problems, which are inherently sequential parameterized problems. These problems are parameterized versions of the well-known bounded halting problem for deterministic Turing machines and the Boolean circuit evaluation problem. This means that there are parameterized problems for which adding more processors provides no significant speedup in (parallel) time required to find a solution. Contrast this parameterized inherent sequentiality with the highly parallelizable parameterized versions of P-complete problems in the previous section. In light of this, we recommend that researchers consider paraP-completeness when determining membership of a parameterized problem in paraP.

4.4.1 Completeness in paraP

We define P-completeness so that problems that are P-complete are unlikely to see a significant decrease in time complexity when parallelism is allowed, under the assumption that NC $\neq$ P. Let us define paraP-completeness similarly, so that paraP-complete problems are unlikely to see a significant decrease in "parameterized" time complexity when "parameterized" parallelism is allowed, under the assumption that paraNC $\neq$ paraP. We already know that each P-complete problem induces a paraP-complete problem with a trivial parameterization [34, Proposition 14], however we are interested in natural problems with non-trivial parameterizations. We prove the existence of nontrivial inherently sequential parameterized problems that are not

in paraNC (under the assumption paraNC $\neq$ paraP) and whose underlying decision problems are P-complete, complementing section 4.3, which proves the existence of parameterized problems in paraNC whose underlying decision problems are P-complete.

Specifically, we show that parameterized versions of the circuit evaluation problem and the bounded halting problem are paraP-complete This means that the classical P-complete problems do in fact become paraP-complete, but it is important to choose the correct parameterization (which is not obvious in some cases). Furthermore, this complements the work of [32], in which the authors show complete problems for several other parameterized complexity classes. These results provide fundamental limits to parallelizability for parameterized problems.

We can really only say that our paraP-complete problems are inherently sequential under the assumption that paraNC $\neq$ paraP. This assumption is reasonable because it is equivalent to the inequality NC $\neq$ P.

Proposition 4.4.2. paraNC = paraP *if and only if* NC = P.

Proof. The proof is trivial if we use the definitions of the complexity class para$\mathcal{C}$ as the class of all parameterized problems (Q, κ) for which there is a language L in the complexity class $\mathcal{C}$ such that $x \in Q$ if and only if $(x, 1^{f(\kappa(x))}) \in L$. See [34, Proposition 8], for example. □

Completeness is defined for paraP with respect to paraNC many-one reductions, in order that a paraNC algorithm for any one paraP-complete problem implies a paraNC algorithm for every paraP problem.

Definition 4.4.3 (paraP-completeness)**.** A parameterized problem (Q, κ) is paraP-*hard* if for each parameterized problem (R, λ), there is a paraNC many-one reduction from (R, λ) to (Q, κ). If furthermore (Q, κ) is in paraP, then it is paraP-*complete*.

Proposition 4.4.4. *If a* paraP*-complete problem is in* paraNC*, then* paraNC = paraP*.*

Proof. Follows from the downward closure of paraNC under paraNC many-one reductions. □

The following problem is adapted from Short Deterministic Turing Machine Computation in [17].

Definition 4.4.5 (p-Bounded Halting Problem, aka p-BHP)**.**

Instance: deterministic Turing machine M, binary string x of length n, positive integer t in unary, positive integer c.

Parameter: t/n^c

Question: Does M accept x within t steps?

Theorem 4.4.6. *p-BHP is* paraP*-complete.*

Proof. The underlying decision problem is in P (by a standard simulation on the deterministic universal Turing machine), so the parameterized problem is in paraP. To show paraP-hardness, we use a generic reduction. Let (Q, κ) be an arbitrary parameterized problem in paraP and let M be the deterministic Turing machine that decides Q in time $f_M(k)n^c$ for some (circuit-)computable function f_M and some positive integer c. The reduction is $x \mapsto (M, x, 1^{f_M(k)n^c}, c)$. This is computable by a (nonuniform) circuit family of constant depth and size $f(k)n^{O(1)}$, where f is a circuit-computable function. The parameter of the reduced instance is $f_M(k)n^c/n^c$, or simply $f_M(k)$, which satisfies the parameter bound required by the definition of paraNC many-one reduction. Therefore we conclude that p-BHP is paraP-complete. □

The following problem is a modification of BS-BD-CVP from [18].

Definition 4.4.7 (p-Small Circuit Evaluation, aka p-SCE)**.**

Instance: Boolean circuit C on n inputs, binary string x of length n, positive integer k, positive integer α, multi-output Boolean circuit f with size and depth of C at most $f(k)n^{\alpha}$.

Parameter: k

Question: Does $C(x) = 1$?

This theorem is related to [18, Corollary 2], where the authors prove that the BS-BD-CVP problem is complete for the class PNC (a class that exists between paraNC and paraP) under paraNC many-one reductions. While their reduction is a generic reduction, ours is a reduction from the parameterized bounded halting problem.

Theorem 4.4.8. *p-SCE is* paraP*-complete.*

Proof. Membership in paraP is straightforward to prove: use the natural algorithm for evaluating a circuit which can be performed in linear time with respect to the size of the circuit. We must also compute $f(k)n^{\alpha}$ and compare it with the size and depth of the circuit C. Both of these are polynomial-time algorithms with respect to the size of the input, and hence the problem is in paraP.

Now we prove paraP-hardness. The reduction from p-BHP is

$$(M, x, 1^t, c) \mapsto (C_M, x, k, \alpha, f),$$

where

- C_M is the standard circuit of size $O(t^2)$ and depth $O(t)$ simulating t steps of the action of M on inputs of length n,
- x is copied directly from the input,

- $k = t/n^c$.
- $\alpha = 2c$,
- f is the function $x \mapsto x^2$,

This reduction is computable in the appropriate size and depth bounds, and its correctness follows from the correctness of the standard deterministic Turing machine-to-circuit reduction. To check that the reduced instance is well-formed, let us verify that the circuit C_M meets the size and depth requirements. The size of C_M is $O(t^2)$, which is $O(((t/n^c)n^c)^2)$, or simply $f(k)n^\alpha$. Similarly the depth of C_M is $O(t)$, which is smaller than $O(t^2)$, and thus bounded above by $f(k)n^\alpha$ as well. (There are some constants in the size and depth bounds that we have ignored, but those can be incorporated into the definition of f.) Finally, the parameter in the original instance, t/n^c, is exactly the parameter of the reduced instance, so this reduction meets the necessary paramater bound. Therefore we have shown a correct paraNC many-one reduction from a paraP-complete problem. □

As expected, if any of these paraP-complete problems are fixed-parameter parallelizable, then paraNC = paraP.

It seems that most P-complete problems will end up being paraP-complete under this notion of completeness. This doesn't really help us distinguish between different P-complete problems based on how fixed-parameter parallelizable they are. In the next subsection, we try a different approach.

4.4.2 Parameterized complexity of efficient verification

There is a nice relationship between the circuit satisfiability (decision) problem and the circuit evaluation problem: an algorithm for the latter is a verification procedure

for the former, given a satisfying assignment. The relationship between these two problems in particular reflect the general relationship between NP and P. Does the same sort of relationship hold for the parameterized versions of these classes, paraWP and paraP? We define a new parameterized complexity class to address this question.

We show that our parameterized complexity class, paraEP, the verification class of paraWP, is a subclass of paraP. We are unable to show that is equal to paraP, so our intuition does not yet match our definitions. What remains is to show a natural problem in this class and to determine whether it equals paraP or not.

We start by considering the parameterized weighted circuit satisfiability and circuit evaluation problems. A circuit is *k-satisfiable* if there is a satisfying assignment of Hamming weight exactly k.

Definition 4.4.9 (p-Circuit k-Satisfiability, aka p-k-CSat)**.**

Instance: Boolean circuit C, natural number k.

Parameter: k.

Question: Is C k-satisfiable?

The corresponding parameterized weighted circuit evaluation problem would then be as follows. Let $\|x\|_1$ denote the Hamming weight (that is, the number of ones) in x.

Definition 4.4.10 (p-Circuit k-Evaluation, aka p-k-CE)**.**

Instance: Boolean circuit C, binary string x, natural number k.

Parameter: k.

Question: Does $\|x\|_1 = k$ and $C(x) = 1$?

Enforcing that the Hamming weight of the string is exactly the parameter in this way is a bit superfluous, since the Hamming weight of x can be computed easily (in NC^1 but not in AC^0), but this problem is technically the "verification"

problem corresponding to the satisfiability problem above. Instead, we use a slightly more natural problem that remains equivalent to this one under paraNC^1 many-one reductions.

Definition 4.4.11 (p-Weighted Circuit Evaluation, aka p-WCE)**.**

Instance: Boolean circuit C, binary string x.

Parameter: $\|x\|_1$.

Question: Does $C(x) = 1$?

In the setting of decision problems, we know that NP can be characterized as the closure of the circuit satisfiability problem under polynomial-time many-one reductions,

$$\mathsf{NP} = [\text{Circuit Satisfiability}]^{\leq_m^{\mathsf{P}}},$$

and P as the closure of the circuit evaluation problem under NC^1 many-one reductions,

$$\mathsf{P} = [\text{Circuit Evaluation}]^{\leq_m^{\mathsf{NC}^1}}.$$

In the setting of parameterized problems, the class paraWP can be characterized as the closure of the parameterized weighted circuit satisfiability problem under fixed-parameter tractable many-one reductions,

$$\mathsf{paraWP} = [p\text{-}k\text{-CSat}]^{\leq_m^{\mathsf{paraP}}}.$$

Following the above pattern, we define a new class as the closure of the parameterized weighted circuit evaluation problem under fixed-parameter parallelizable many-one reductions,

$$\mathsf{paraEP} = [p\text{-WCE}]^{\leq_m^{\mathsf{paraNC}}}$$

("E" for evaluation).

Since the underlying decision problem, the problem of evaluating a circuit on a given input, is in P, the parameterized problem p-WCE is trivially in paraP. Since paraNC reductions compose, paraNC is a subset of paraP, and paraP is closed under paraP reductions, we conclude that paraEP is a subset of paraP.

Theorem 4.4.12. paraEP $\subseteq$ paraP.

Is paraEP = paraP? The standard simulation of a deterministic Turing machine by a circuit, as in [52], for example, fails to provide a paraNC many-one reduction to the parameterized weighted circuit evaluation problem, since the natural reduction would be of the form $x \mapsto (C, x, \|x\|_1)$, but the parameter value $\|x\|_1$ is not necessarily bounded by a function of $\kappa(x)$. The same issue prevents us from showing that paraNC $\subseteq$ paraEP.

4.5 paraNC is to NC as paraWNC is to NNC

The previous section shows one way of proving that a problem is likely not parallelizable, even in a parameterized sense: proving it complete for paraP. A highly parallel algorithm for such a problem would imply that every problem that could be solved by a fixed-parameter tractable algorithm could be solved by a fixed-parameter parallelizable algorithm, a notion that violates our intuition about the nature of time. There is another way of proving a problem unlikely to be fixed-parameter parallelizable, one that relies on our intuition about the nature of nondeterminism. Our intuition is that nondeterministic computation cannot be simulated deterministically by any algorithm that is significantly more efficient than simply enumerating each possible branch of the nondeterministic computation. We can use this intuition to provide evidence that an entire family of weighted circuit satisfiability problems is unlikely to be parallelizable

in a parameterized sense; this section formalizes this idea.

Specifically, Theorem 4.5.15 proves that $\mathsf{NC}^d = \mathsf{NNC}^d[\omega(\log n)]$ is equivalent to the corresponding collapse of the classes of parameterized problems. On the way to proving this equivalence, we also prove that the parameterized versions of the natural complete problems in $\mathsf{NNC}^d[\omega(\log n)]$ are complete for the corresponding parameterized complexity classes (Theorem 4.5.11). Before getting to those theorems, we of course define the necessary parameterized complexity classes and provide some examples of member problems. These complete problems and complexity class collapses validate our intuition that the parameterized complexity classes behave like the classes of problems decidable by algorithms augmented with limited nondeterminism. These results complement similar equivalences from [35, Theorem 3.29] and [19, Theorem 15], as discussed in subsection 4.5.4 below. We stop short of providing a general theorem that supercedes all of these theorems.

4.5.1 Definition of paraWNC

The paraW "operator" defined in [32, Definition 3.1] applies generically to an arbitrary complexity class as follows. If $\mathcal{C}$ is a class of decision problems, then para$\mathsf{W}\mathcal{C}$ is the class of parameterized problems (Q, κ) for which there is a $\mathcal{C}$ machine M and computable functions f and h such that x in Q if and only if there is a string w of length $h(\kappa(x)) \log |x|$ such that M accepts on input $(x, 1^{f(\kappa(x))})$ and nondetermistic input w; access to the nondeterministic input is two-way. In a loose sense, for most classes $\mathcal{C}$, deterministic $\mathcal{C}$ is to nondeterministic $\mathcal{C}$ as para$\mathcal{C}$ is to para$\mathsf{W}\mathcal{C}$.

When $\mathcal{C} = \mathsf{NC}$ in particular, we get the following equivalent definition.

Definition 4.5.1 (paraWNC^d)**.** Let d be a natural number. A parameterized problem (Q, κ) is in the class paraWNC^d if there are circuit-computable functions f and h, and

a nondeterministic circuit family $\{C_{n,k}\}$ such that for each string x,

- $x \in Q$ if and only if $C_{n,k}(x) = 1$, where $n = |x|$ and $k = \kappa(x)$,
- $\text{size}(C_{n,k}) \leq f(k)n^{O(1)}$,
- $\text{depth}(C_{n,k}) \leq f(k) + \log^d n$,
- $\text{nondet}(C_{n,k}) \leq h(k) \log n$.

Proposition 4.4.2 already showed us that $\mathsf{paraNC} = \mathsf{paraP}$ if and only if $\mathsf{NC} = \mathsf{P}$. We conjecture that a similar equivalence holds for the nondeterministic versions of these classes as well.

Conjecture 4.5.2. *Suppose d is a positive integer.* $\mathsf{paraWNC}^d = \mathsf{paraWP}$ *if and only if there is a nondecreasing, unbounded, circuit-computable function i such that* $\mathsf{NNC}^d[i(n) \log n] = \mathsf{NP}[i(n) \log n]$.

4.5.2 Example problems in paraWNC

What kind of problems are in the class paraWNC? Some problems in $\mathsf{paraWNC}^1$ are given in [32]; we provide a few more problems in each class $\mathsf{paraWNC}^d$. This section exhibits parameterized problems that are in $\mathsf{paraWNC}^1$, $\mathsf{paraWNC}^2$, and, more generally, $\mathsf{paraWNC}^d$ for each positive integer d.

Specifically, we prove that the group rank problem is in paraWL, a subset of $\mathsf{paraWNC}^2$. We also prove that the weighted NC^d circuit satisfiability problem is in $\mathsf{paraWNC}^d$ and give an alternate proof that the weighted formula satisfiability problem is in $\mathsf{paraWNC}^1$. Along with the membership of the weighted circuit satisfiability problem in paraWP, we now have an example problem in each parameterized complexity

class in the inclusion chain

$$\mathsf{paraWNC}^1 \subseteq \mathsf{paraWNC}^2 \subseteq \cdots \subseteq \mathsf{paraWP}.$$

Small generating set problems

The parameterized semigroup rank problem is in paraWNL [32, Theorem 3.12], which is contained in $\mathsf{paraWNC}^2$. The group rank problem, a restricted version of the semigroup rank problem, is useful in applications and generally lacks an efficient implementation, so we would like to understand the parameterized complexity of that problem as well. For example, at the time of this publication, the popular computational discrete algebra software package GAP [37] includes a function `RankPGroup` that computes the rank of a p-group but lacks a function that computes the rank of a general finite group.

Definition 4.5.3 (p-Group Rank)**.**

Instance: finite group G given as a product table, positive integer k.

Parameter: k.

Question: Does G have a generating set of cardinality k?

We show that the problem is in paraWL. Although we are unable to show that the problem is in $\mathsf{paraWNC}^1$, membership in paraWL still implies membership in $\mathsf{paraWNC}^2$, so for this problem, parameterized verification is considered highly parallel. The high-level algorithm is quite simple: nondeterministically choose a subset and verify that the subset generates each element of the group.

Theorem 4.5.4. *p-Group Rank is in* paraWL*.*

Proof. The Turing machine receives G and k as input and a subset S of k group

elements as a witness. It loops over each element g in G and decides whether $g \in \langle S \rangle$. The machine accepts exactly when all of the subgroup membership tests pass. Looping over n elements uses $O(\log n)$ space. Deciding whether g is in $\langle S \rangle$ is the subgroup membership problem, which is in SL, which in turn equals L. Thus the overall space usage is $O(\log n)$. The size of the witness is $k \log n$, and two-way access is required, since we execute the subgroup membership procedure n times. We conclude that this parameterized problem is in paraWL. □

Weighted circuit satisfiability problems

We know that p-FSAT is in $\mathsf{paraWNC}^1$ and that p-CSAT is in paraWP. We would like to show that there problems in each parameterized class $\mathsf{paraWNC}^d$ between $\mathsf{paraWNC}^1$ and paraWP. To this end, we interpolate between Boolean formulas and Boolean circuits to construct parameterized weighted satisfiability problems. This section demonstrates that interpolation and proves membership of each problem in $\mathsf{paraWNC}^d$.

We show that the problem of deciding whether a $\log^d n$ depth circuit is satisfiable by an input of Hamming weight k is in $\mathsf{paraWNC}^d$ when parameterized by k. These are the first defined problems in these parameterized complexity classes. In a later section, we will show that these problems are complete as well.

Let $\|x\|_1$ denote the Hamming weight (that is, the number of ones) of a binary string. This problem is the restriction of p-CSAT to bounded depth circuits, thus we expect it to be of lower computational complexity.

Definition 4.5.5 (p-k-NC^dCSAT)**.**

Instance: Boolean circuit C on m inputs with depth $\log^d m$, natural number k.

Parameter: k.

Question: Is there a binary string w such that $\|w\|_1 = k$ and $C(x) = 1$?

To prove membership in $\mathsf{paraWNC}^d$, we will use the following *decoder function*. Since we will also later use its inverse, the *encoder function*, we define it here as well.

Definition 4.5.6. The *encoder function*, $E_n \colon \{0,1\}^m \to \{0,1\}^{\log m}$, is defined by

$$E_m(x) = \begin{cases} i & \text{if } x \text{ has exactly one 1 at index } i \\ 0^{\log m} & \text{otherwise.} \end{cases}$$

for each binary string x of length m. The *decoder function*, D_m, is defined as the inverse of the encoder function.

Lemma 4.5.7. *For each natural number m, both the encoder and decoder functions are computable by a circuit of size $O(m)$ and depth $O(\log m)$.*

Proof. This can be found in, for example, Lemmas 2.5.3 and 2.5.4 of [58]. □

This theorem is an adaptation of [15, Lemma 3.3].

Lemma 4.5.8. *For each positive integer d, we have p-k-$\mathsf{NC}^d CSAT$ is in $\mathsf{paraWNC}^d$.*

Proof. The algorithm will be composed of several subcircuits.

- Let U be the depth-universal circuit [20] for depth $\log^d m$.
- Let $D_{m,k}$ be the function defined by $D_{m,k}(w) = D_m(w_1) \vee \cdots \vee D_m(w_k)$, where $\vee$ denotes bitwise OR for strings of length m, the binary string w_i is the ith block of w of size $\log m$, and D_m is the decoder function (Definition 4.5.6).

- Let Δ be the function that takes k blocks of $\log m$ bits each as input and evaluates to true exactly when each pair of blocks of size $\log m$ are distinct. For example, if $m = 4$, then $\Delta(0101) = 0$ but $\Delta(1011) = 1$.

Define the nonuniform, nondeterministic NC^d circuit family $\{A_{m,k}\}$ that decides p-k-NC^dCSAT by

$$A_{n,k}((C,k),w) = U(C, D_{m,k}(w)) \wedge \Delta(w).$$

In other words, $A_{n,k}$ interprets its witness string w of length $k \log m$ as an encoding of a string of length m containing *exactly* k ones (as enforced by Δ), then evaluates the circuit C on that string.

This algorithm correctly decides the underlying decision problem. For each circuit C of size n and each integer k,

$$\begin{aligned}
(C,k) \in p\text{-}k\text{-}\mathsf{NC}^d\text{CSAT} &\iff \exists x \in \{0,1\}^m \colon \|x\|_1 = k \text{ and } C(x) = 1 \\
&\iff \exists w \in \{0,1\}^{k \log m} \colon (C(D_{m,k}(w)) \wedge \Delta(w)) = 1 \\
&\iff \exists w \in \{0,1\}^{k \log m} \colon (U(C, (D_{m,k}(w))) \wedge \Delta(w)) = 1 \\
&\iff \exists w \in \{0,1\}^{k \log m} \colon A_{n,k}((C,k),w) = 1.
\end{aligned}$$

If we assume without loss of generality that $m \leq n$ (by padding with useless gates, for example), then the number of nondeterministic bits used is less than $k \log m$, which is of the form $h(k) \log n$.

For the size and depth bounds, we need to determine the size and depth of the circuits for U, $D_{m,k}$, and Δ.

- The depth-universal circuit U has size $m^{O(1)}$ and depth $O(\log^d n)$, which is $O(\log^d m)$ because we can assume without loss of generality that the size of C is at least the number of its inputs.

- Decoding a single block of size $\log m$ requires size $O(m)$ and depth $O(\log m)$ by Lemma 4.5.7. Decoding all k blocks of size $\log m$ requires k copies of that subcircuit, along with an OR tree for each of the m output bits. Thus the size of $D_{m,k}$ is $O(km + m \log k)$ and the depth $O(\log m + \log k)$.

- Comparing two binary strings of length $\log m$ for inequality requires $O(\log m)$ size and $O(\log \log m)$ depth. Comparing all $\binom{k}{2}$ pairs of blocks and requiring they are all distinct thus requires a circuit of size $O(\binom{k}{2} \log m)$ and depth $O(\log m + \log k)$.

Hence the overall size of $A_{n,k}$ is of the form $f(k)m^{O(1)}$ and the depth $f(k) + \log^d m$. Since we can assume without loss of generality that $m \leq n$, we get size and depth bounds with the appropriate dependence on k and n.

Since we have shown a $\mathsf{paraWNC}^d$ circuit family deciding p-k-NC^dCSAT, we conclude that the problem is in $\mathsf{paraWNC}^d$. □

If we replace k with any function $i(n)$ bounded above by a polynomial in n, we get the (already known) membership of the underlying decision problem in $\mathsf{NNC}^d[i(n) \log n]$. The weight k can be at most the number of inputs to the circuit, which is in turn at most the size of the circuit, so a polynomial upper bound on k suffices to cover all meaningful values of k.

Corollary 4.5.9 ([15]). *Suppose d is a positive integer and i is a circuit-computable function such that $i(n) \leq n^{O(1)}$. Then the decision problem $i(n)$-NC^d-CSAT is in nonuniform $\mathsf{NNC}^d[i(n) \log n]$.*

Proof. The proof is similar to that of Lemma 4.5.8 with only one addition necessary to the algorithm: deciding whether $k \leq i(n)$. To do this, add a single AND gate whose first input is $A_{n,k}((C, k), w)$ and whose second input is the result of deciding the inequality

$k \leq i(n)$. The computation of $i(n)$ can be assumed part of the nonuniformity of the circuit, at which point the comparison requires only logarithmic depth. (If we wish the deciding circuit to be uniform, we can require that $i(n)$ be computable in uniform depth $O(\log^d n)$.) □

Weighted formula satisfiability problems

Lemma 4.5.8 demonstrates a parameterized highly parallel algorithm for certain circuit satisfiability problems. This technique can be adapted to work for a simpler computational model, Boolean formulas, as well. In fact, this provides an alternate proof that parameterized weighted Boolean formula satisfiability problem, denoted p-FSAT, is in $\mathsf{paraWNC}^1$, implicit in [32, Theorem 3.6]. We adapt the components of the algorithm from circuits to formulas.

Specifically, we show how to implement the decoder function of Definition 4.5.6 as a Boolean formula, and how that can be used to decide whether a Boolean formula has a satisfying assignment of a given weight. This is interesting because we have used an algorithm designed for circuits on formulas in an unexpected way.

Lemma 4.5.10 ([32, Theorem 3.6])**.** *p-FSAT is in $\mathsf{paraWNC}^1$.*

Proof. We adapt the "$k \log n$ trick" from circuit inputs to formula variables. This requires a reimplementation of the decoder function of Definition 4.5.6 as a function on Boolean variables. Similar to the proof of Lemma 4.5.8, the algorithm involves composing a decoder and an algorithm for evaluating a Boolean formula, after nondeterministically choosing a witness. Again, we use some subcircuits:

- Let U be the NC^1 algorithm for evaluating a Boolean formula [12, 13].
- Let M_i be the function on $\log m$ inputs that outputs the ith minterm of its input

variables (for example, $M_4(w_1, w_2, w_3) = w_1 \wedge \neg w_2 \wedge \neg w_3$). This function acts like the decoder in Lemma 4.5.8.

- Let $R_{m,k}(\phi)$ be the function that replaces each instance of a variable x_i in a Boolean formula ϕ on m variables with $\vee_{j=1}^{k} M_i(\vec{v}_j)$, where $\vec{v}_j$ denotes the jth block of size $\log m$ in a tuple of $k \log m$ new variables $v_1, \ldots, v_{k \log m}$.

- Let Δ be the function that takes k blocks of $\log m$ bits each as input and evaluates to true exactly when each pair of blocks of size $\log m$ are distinct.

Define the nonuniform, nondeterministic NC^1 circuit family $\{A_{n,k}\}$ that decides p-FSAT by

$$A_{n,k}((\phi, k), w) = U(R_{m,k}(\phi), w) \wedge \Delta(w).$$

In other words, $A_{n,k}$ interprets its witness string w of length $k \log m$ as the *encoding of an assignment* to the variables of ϕ in which exactly k variables are set to true, then evaluates the formula ϕ with respect to the decoded assignment.

This algorithm correctly decides the weighted Boolean formula satisfiability problem. Similar to the proof of Lemma 4.5.8, ϕ has a satisfying assignment of weight exactly k if and only if ϕ' has a satisfying assignment (of arbitrary weight). If we assume without loss of generality that $m \leq n$ (by padding with tautological conjuncts, for example), then the number of nondeterministic bits used is less than $k \log m$, which is of the form $h(k) \log n$.

The algorithm also has the appropriate size and depth bounds. Since each minterm is of size exactly $\log m$ and each variable x_i is represented the disjunction of k such minterms, the new formula ϕ' is of size $|\phi| k \log m$, which is just $nk \log m$.

- Each variable x_i can be replaced in parallel, and within that replacement, each disjunct $M_i(\vec{w}_j)$ can be replaced in parallel as well. The circuit that writes

$M_i(\vec{w}_j)$ is a circuit of size $O(\log m)$ and depth $O(\log m)$ (for using the index i as a selector in a multiplexer), so $R_{m,k}$ can be implemented by a circuit of size $O(nk \log m)$ and depth $O(\log m)$.

- The circuit U receives a formula of size $O(nk \log m)$ and an assignment of length $O(k \log m)$. Since U is a circuit of size polynomial in its the size of its input, its size is $O((nk \log m)^c)$ for some constant c. Similarly, its depth is $O(\log^d(nk \log m))$.

- As in Lemma 4.5.8, the circuit for Δ is of size $O(\binom{k}{2} \log m)$ and depth $O(\log m + \log k)$.

The overall size and depth of the circuit $A_{n,k}$ are therefore of the form $f(k)n^{O(1)}$ and $f(k) + O(\log^d n)$, respectively.

At this point, we have shown a correct $\mathsf{paraWNC}^1$ algorithm for p-FSAT. □

The same proof works for the problems of deciding whether a circuit or a formula has a satisfying assignment of weight *at most* k, as well (one could even remove the Δ subcircuit entirely, but that is not necessary).

4.5.3 Completeness in paraWNC

We saw that the parameterized weighted Boolean formula satisfiability problem, p-FSAT, is in $\mathsf{paraWNC}^1$ in Lemma 4.5.10. In fact, it is complete for $\mathsf{paraWNC}^1$ under paraFO many-one reductions [32, Theorem 3.6]. The parameterized weighted Boolean circuit satisfiability problem, denoted p-CSAT, is the same problem with Boolean circuits instead of Boolean formulas; this problem is complete for paraWP under paraFO many-one reductions by a similar proof. It makes sense to expect, then, that for each positive integer d the parameterized problem p-k-NC^dCSAT may be complete

for the class $\mathsf{paraWNC}^d$. We adapt the strategy used to prove completeness of p-CSAT to prove completeness of p-k-NC^dCSAT. This section describes the strategy and shows how it applies to both p-k-NC^dCSAT and p-FSAT.

Specifically, we use the strategy from [15, Theorem 3.6], which relies on the "$k \log n$ trick" (see also [35, Corollary 3.13], or the origin [1]). There the authors prove only that the decision problem underlying p-k-NC^dCSAT is complete for $\mathsf{NNC}^d[k \log n]$. Now we have a full interpolation for the inclusion chain

$$\mathsf{paraWNC}^1 \subseteq \mathsf{paraWNC}^2 \subseteq \cdots \subseteq \mathsf{paraWP},$$

via the chain of parameterized reductions between corresponding complete problems

$$p\text{-FSAT} \leq p\text{-}k\text{-}\mathsf{NC}^2\text{CSAT} \leq \cdots \leq p\text{-CSAT}.$$

Whether the p-k-NC^dCSAT problems are complete under paraFO many-one reductions remains open. In any case, as we will see in the next section (and repeatedly throughout this paper), parameterized complexity and limited nondeterminism in decision complexity are closely related.

Theorem 4.5.11. *For each positive integer d, we have p-k-NC^dCSAT is complete for $\mathsf{paraWNC}^d$ under paraNC^1 many-one reductions.*

Proof. Membership in $\mathsf{paraWNC}^d$ was proven in Lemma 4.5.8.

Suppose (Q, κ) is in $\mathsf{paraWNC}^d$, so there is a nonuniform NC^d circuit family $\{C_{n,k}\}$ and circuit-computable functions f and h such that

- $x \in Q$ if and only if $C_{n,k}(x) = 1$,
- $\text{size}(C_{n,k}) \leq f(k)n^{O(1)}$,

- $\text{depth}(C_{n,k}) \leq f(k) + O(\log^d n)$,
- $\text{nondet}(C_{n,k}) \leq h(k) \log n$.

On input x of length n, let C_x denote the circuit $C_{n,k}$ with x hardcoded as its first n inputs. Thus C_x is a circuit with $h(k) \log n$ inputs such that $x \in Q$ if and only if C_x is satisfiable. Let $E_{n,k}$ denote the function defined by $E_{n,k}(w) = E_n(w_1) \circ \cdots \circ E_n(w_{h(k)})$, where $\circ$ denotes string concatenation, E_n is the encoder function of Definition 4.5.6, and w_i is the ith block of size n in the string w, for each string w of length $h(k)n$. The reduction is then $x \mapsto (C', h(k))$, where $C' = C_x \circ E_{n,k}$.

The circuit C' is of the correct form to be an input to p-k-NC^dCSAT. The size of C' is

$$\begin{aligned}
\text{size}(C') &= \text{size}(E_{n,k}) + \text{size}(C_x) \\
&\leq h(k)\,\text{size}(E_n) + f(k)n^{O(1)} \\
&\leq h(k)O(n) + f(k)n^{O(1)}
\end{aligned}$$

and the depth

$$\begin{aligned}
\text{depth}(C') &= \text{depth}(E_{n,k}) + \text{depth}(C_x) \\
&\leq \text{depth}(E_n) + (f(k) + \log^d n) \\
&\leq O(\log n) + f(k) + \log^d n \\
&\leq f(k) + O(\log^d n).
\end{aligned}$$

The number of inputs to C' is $h(k)n$, so we need the size to be polynomial in $h(k)n$ and the depth to be $\log^d(h(k)n)$. The size bound is satisfied if we choose $h(k)$ so that $f(k) \leq h(k)$ and the depth bound satisfied if we choose $h(k)$ so that $f(k) \leq \log^d h(k)$,

ignoring some constants that can be incorporated into the function h. We can choose h this way without loss of generality, because choosing a larger h does not affect membership of (Q, κ) in $\mathsf{paraWNC}^d$. (This does, however, cause the nondeterminism upper bound of the problem (Q, κ) to be extremely loose and the dependence on k in the size and depth bounds of C' to be extremely high, but it is technically sufficient.)

The reduction is a correct many-one reduction between the underlying decision problems. Suppose $x \in Q$, so C_x is satisfiable. Since E_n is surjective, so is $E_{n,k}$, hence there is a string w (of length $h(k)n$) such that $C'(w) = 1$. Furthermore, the number of ones in the string equals $h(k)$, or in other words $\|w\|_1 = h(k)$, since all preimages of $E_{n,k}$ satisfy this equality.[1] Therefore, C' has a satisfying assignment of Hamming weight exactly $h(k)$. For the converse, suppose C' has a satisfying assignment w of weight exactly $h(k)$. Then there is a satisfying assignment of length $h(k) \log n$ for C_x, namely $E_{n,k}(w)$. Therefore, x is in Q.

The reduction is paraNC^1-computable. The size of the circuit computing the reduction is simply the size of the output, which is $\text{size}(C') + \text{size}(h_k)$, where h_k denotes the circuit computing h on inputs of size k. Both addends are of the form $f'(k)n^{O(1)}$ for some circuit-computable function f'. The depth of the circuit computing the reduction is dominated by the depth of the h_k, which is bounded above by $f'(k) + \log n$ for some function f'. Thus the size and depth requirements for the reduction are met. Finally, if R denotes the reduction and κ' denotes the parameterization for p-k-$\mathsf{NC}^d\text{CSAT}$,

$$\kappa'(R(x)) \leq \kappa'((C', h(\kappa(x)))) = h(\kappa(x)).$$

Since h is circuit-computable by hypothesis, the reduction meets the parameterization

[1] Technically, we need to guarantee the satisfying input to C_x has no all-zero blocks to make this statement. For each parameterized problem (Q, κ) there is another equivalent problem that satisfies such a requirement on the witnesses for Q with no change in complexity.

bound.

Since we have shown a correct paraNC^1 many-one reduction from an arbitrary parameterized problem in $\mathsf{paraWNC}^d$ to p-k-NC^dCSAT, we conclude that p-k-NC^dCSAT is complete for the class. □

As in Corollary 4.5.9, if we replace k with any function $i(n)$ bounded above by a polynomial in n, we get the (already known) completeness of the decision problem $i(n)$-NC^d-CSAT in the complexity class $\mathsf{NNC}^d[i(n)\log n]$. This is a slight improvement, since [15, Theorem 3.6] proves that the problem of deciding whether a circuit has a satisfying assignment of weight at most $i(n)$ is complete for $\mathsf{NNC}^d[i(n)\log n]$ for all $d \geq 2$ under logarithmic space many-one reductions.

Corollary 4.5.12 ([15, Theorem 3.6])**.** *Suppose d is a positive integer and i is a circuit-computable function such that $i(n) \leq n^{O(1)}$. Then $i(n)$-NC^d-CSAT is complete for $\mathsf{NNC}^d[i(n)\log n]$ under nonuniform NC^1 many-one reductions.*

Proof. Membership was proven in Corollary 4.5.9. The proof of completeness is identical to that of the previous theorem, replacing k with $i(n)$. (If we wish to have uniform completeness, we can require that $i(n)$ be computable in uniform depth $O(\log n)$.) □

Finally, we extend [32, Corollary 3.7] using this new family of complete problems. That theorem states that $\mathsf{NC}^1 = \mathsf{P}$ implies $\mathsf{W[SAT]} = \mathsf{W[P]}(= \mathsf{paraWP})$, where $\mathsf{W[SAT]}$ is the closure of p-FSAT under paraP many-one reductions and $\mathsf{W[P]}$ is the closure of p-CSAT. We wish to generalize this to allow for an interpolation between NC^1 and P. This connects a collapse in parameterized complexity classes to one in classical complexity classes.

Corollary 4.5.13. *For each positive integer d, if* $\mathsf{NC}^d = \mathsf{P}$, *then* $\mathsf{W}[\mathsf{NC}^d\mathsf{SAT}] = \mathsf{W}[\mathsf{P}](= \mathsf{paraWP})$.

Proof. By definition, $\mathsf{W}[\mathsf{NC}^d\mathsf{SAT}]$ is the closure of p-k-NC^dCSAT under paraP many-one reductions and $\mathsf{W}[\mathsf{P}]$ is the closure of p-CSAT under paraP many-one reductions. If $\mathsf{NC}^d = \mathsf{P}$, then there is a paraP many-one reduction from p-CSAT to p-k-NC^dCSAT (because p-CSAT is now in NC^d). Thus every parameterized problem that reduces to p-CSAT also reduces to p-k-NC^dCSAT. We conclude that $\mathsf{W}[\mathsf{NC}^d\mathsf{SAT}] = \mathsf{W}[\mathsf{P}]$. □

4.5.4 Does paraNC equal paraWNC?

We now have complete problems in the parameterized complexity classes that are the analogs of the nondeterministic NC hierarchy. With complete problems, the guiding question now becomes whether a deterministic simulation of these nondeterministic classes causes a complexity theoretic collapse elsewhere. We demonstrate how a collapse in parameterized complexity classes is equivalent to a collapse in classical complexity classes.

The main theorem in this section, Theorem 4.5.15, proves that $\mathsf{paraNC} = \mathsf{paraWNC}$ if and only if NC can deterministically simulate $\omega(\log n)$ nondeterministic bits. Compare it with the following similar theorems (here β indicates one-way access to the witness as opposed to W, which indicates two-way access).

- $\mathsf{paraP} = \mathsf{paraWP}$ if and only if $\mathsf{P} = \mathsf{NP}[\omega(\log n)]$ [35, Theorem 3.29].
- $\mathsf{paraL} = \mathsf{para}\beta\mathsf{L}$ if and only if $\mathsf{L} = \mathsf{NL}[\omega(\log n)]$ [19, Theorem 15].

If you believe that nondeterminism is hard to simulate deterministically, then this provides evidence that the parameterized complexity classes paraNC and paraWNC are distinct. We conjecture that these theorems can be generalized to generic theorem that subsumes all of them.

This lemma is an adaptation of [35, Lemma 3.24]. It is the "opposite" of Lemma 4.3.5.

Lemma 4.5.14. *Suppose d is a positive integer. For each increasing and circuit-computable function f, there is a function $i_{f,d}$ such that*

$$f(i_{f,d}(n)) \leq \min(n, \log^d n)$$

for each $n \geq f(1)$. Furthermore, $i_{f,d}$ is nondecreasing, unbounded, and circuit-computable. (We call $i_{f,d}$ the "lower inverse" of f.)

Proof. Define $i_{f,d}$ by

$$i_{f,d}(n) = \max\{j \in \mathbb{N} \mid f(j) \leq \min(n, \log^d n)\}.$$

For the boundary case where n is smaller than $f(1)$, define $i_{f,d}(n) = 1$. It is straightforward to prove that this function is nondecreasing and unbounded. (The integer d must be greater than zero to guarantee $i_{f,d}$ is unbounded.)

To compute $i_{f,d}$, we use the fact that f is increasing to perform a binary search on the values $f(1), \ldots, f(\min(n, \log^d n))$ to determine the largest j such that $f(j) \leq \min(n, \log^d n)$. In the worst case, there will be at most $\log n$ comparison subcircuits, each requiring a computation of f, so the overall depth of the circuit computing $i_{f,d}$ is $O(\text{depth}(f) \log n)$ and the size is $O(\text{size}(f) \log n)$. □

This theorem is an adaptation of [35, Theorem 3.29].

Theorem 4.5.15. *Suppose d is a positive integer.* $\mathsf{paraNC}^d = \mathsf{paraWNC}^d$ *if and only if there is a circuit-computable, nondecreasing, unbounded function i such that* $\mathsf{NC}^d = \mathsf{NNC}^d[i(n) \log n]$.

Proof. First we prove the reverse implication. Assume $\mathsf{NC}^d = \mathsf{NNC}^d[i(n)\log n]$, for some i. Since p-k-NC^dCSAT is complete for $\mathsf{paraWNC}^d$ by Theorem 4.5.11, it suffices to show that this problem is in paraNC^d, which necessitates the claimed collapse. By Corollary 4.5.12, the problem $i(n)$-NC^d-CSAT is in $\mathsf{NNC}^d[i(n)\log n]$, which means it is also in NC^d by assumption. If (Q, κ) denotes p-k-NC^dCSAT and $i(n)$-Q denotes $i(n)$-NC^d-CSAT, then the former is in paraNC^d by Corollary 4.3.8.

Now we prove the forward implication. Assume $\mathsf{paraNC}^d = \mathsf{paraWNC}^d$. By Theorem 4.5.11, the parameterized problem p-k-NC^dCSAT is in $\mathsf{paraWNC}^d$, which means it is also in paraNC^d by assumption. Suppose the circuit family witnessing its membership in paraNC^d has size $f(k)n^{O(1)}$ and depth $f(k) + \log^d n$ for some circuit-computable function f. Assume without loss of generality that f is increasing. Choose i to be the "lower inverse" function $i_{f,d}$ guaranteed by Lemma 4.5.14. Since $i(n)$-NC^d-CSAT is complete for $\mathsf{NNC}^d[i(n)\log n]$ by Corollary 4.5.12,it suffices to show that this problem is in NC^d, which necessitates the claimed collapse. If (Q, κ) denotes p-k-NC^dCSAT and $i(n)$-Q denotes $i(n)$-NC^d-CSAT, then the latter is in NC^d by Corollary 4.3.10. □

As expected, algorithms for complete problems are equivalent to collapsing classes.

Corollary 4.5.16. *Suppose d is a positive integer. The following are equivalent.*

1. *p-k-NC^dCSAT is in paraNC^d.*
2. *There is a circuit-computable, nondecreasing, unbounded function i such that $i(n)$-NC^d-CSAT is in NC^d.*
3. *$\mathsf{paraNC}^d = \mathsf{paraWNC}^d$.*
4. *There is a circuit-computable, nondecreasing, unbounded function i such that $\mathsf{NC}^d = \mathsf{NNC}^d[i(n)\log n]$.*

The special case of Theorem 4.5.15 in which the function i is polylogarithmic in n provides further evidence that $\mathsf{NC} \neq \mathsf{NNC}[\mathsf{polylog}]$. This complements Theorem 2.3.5, however this corollary applies only to nonuniform NC circuits.

Corollary 4.5.17. *If* $\mathsf{NC} = \mathsf{NNC}[\mathsf{polylog}]$*, then* $\mathsf{paraNC} = \mathsf{paraWNC}$*.*

4.5.5 Is paraWNC in paraP?

The previous section demonstrates that it is unlikely that $\mathsf{paraWNC}$ is contained in paraNC. This leaves open the possibility that $\mathsf{paraWNC}$ is contained in a larger (but still deterministic) class. By relaxing the collapses in Theorem 4.5.15, we show that it is also unlikely that $\mathsf{paraWNC}$ is contained in paraP. However, the collapses in this theorem are weaker. This is as we expect: a less efficient deterministic simulation of nondeterminism causes less severe consequences in classical complexity theory. Though we do not show it here, this idea can certainly be generalized as well.

Theorem 4.5.18. *Suppose d is a positive integer.* $\mathsf{paraWNC}^d \subseteq \mathsf{paraP}$ *if and only if there is a circuit-computable, nondecreasing, unbounded function i such that* $\mathsf{NNC}^d[i(n) \log n] \subseteq \mathsf{P}$*.*

Proof. The proof is identical to that of Theorem 4.5.15, replacing paraNC^d and NC^d with paraP and P, respectively. It uses versions of Corollary 4.3.8 and Corollary 4.3.10 with similar changes. □

An alternate approach to showing that $\mathsf{paraWNC}^d \subseteq \mathsf{paraP}$ is unlikely is to demonstrate related collapses in parameterized complexity classes. We extend [32, Corollary 3.8], which states that $\mathsf{paraWNC}^1 \subseteq \mathsf{paraP}$ if and only if $\mathsf{paraP} = \mathsf{W}[\mathsf{SAT}]$. As before, we wish to generalize this equivalence to allow for an interpolation between NC^1 and P.

Theorem 4.5.19. *Suppose d is a positive integer.* $\mathsf{paraWNC}^d \subseteq \mathsf{paraP}$ *if and only if* $\mathsf{paraP} = \mathsf{W}[\mathsf{NC}^d\mathsf{SAT}]$.

Proof. If $\mathsf{paraWNC}^d \subseteq \mathsf{paraP}$, then p-k-$\mathsf{NC}^d\mathrm{CSAT}$ is in paraP, so the closure of p-k-$\mathsf{NC}^d\mathrm{CSAT}$ under paraP many-one reductions is contained in the closure of paraP under the same reductions, which is just paraP. Thus $\mathsf{paraP} = \mathsf{W}[\mathsf{NC}^d\mathsf{SAT}]$.

If $\mathsf{paraP} = \mathsf{W}[\mathsf{NC}^d\mathsf{SAT}]$, then every problem that reduces to p-k-$\mathsf{NC}^d\mathrm{CSAT}$ under paraP many-one reductions is in paraP. Since p-k-$\mathsf{NC}^d\mathrm{CSAT}$ is complete for $\mathsf{paraWNC}^d$ under paraNC^d many-one reductions, every problem in $\mathsf{paraWNC}^d$ reduces to p-k-$\mathsf{NC}^d\mathrm{CSAT}$ under paraNC^d many-one reductions, and hence under paraP many-one reductions as well. Thus $\mathsf{paraWNC}^d \subseteq \mathsf{paraP}$. □

Bibliography

[1] Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. "Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues". In: *Annals of Pure and Applied Logic* 73.3 (1995), pp. 235–276. ISSN: 0168-0072. DOI: 10.1016/0168-0072(94)00034-Z.

[2] Richard Anderson and Ernst W. Mayr. *A P-complete problem and approximations to it*. Tech. rep. Stanford, CA, USA: Stanford University, Department of Computer Science, Sept. 1984.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. 1st ed. Cambridge University Press, 2009. ISBN: 0521424267.

[4] Sanjeev Arora and Shmuel Safra. "Probabilistic Checking of Proofs: A New Characterization of NP". In: *Journal of the ACM* 45.1 (Jan. 1998), pp. 70–122. ISSN: 0004-5411. DOI: 10.1145/273865.273901.

[5] S. Arora et al. "Proof verification and hardness of approximation problems". In: *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*. Oct. 1992, pp. 14–23. DOI: 10.1109/SFCS.1992.267823.

[6] G. Ausiello, A. D'Atri, and Marco Protasi. "Lattice theoretic ordering properties for NP-complete optimization problems". In: *Annales Societatis Mathematicae Polonae* 4 (1981), pp. 83–96.

[7] Giorgio Ausiello et al. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. ISBN: 9783540654315.

[8] Max Bannach, Christoph Stockhusen, and Till Tantau. "Fast Parallel Fixed-Parameter Algorithms via Color Coding". In: *arXiv* (2015). URL: http://arxiv.org/abs/1509.06984. Preprint.

[9] Eli Ben-Sasson et al. "On the Concrete Efficiency of Probabilistically-checkable Proofs". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*. Palo Alto, California, USA: ACM, 2013, pp. 585–594. DOI: 10.1145/2488608.2488681.

[10] Ronald V. Book. "Translational lemmas, polynomial time, and $(\log n)^j$-space". In: *Theoretical Computer Science* 1.3 (Feb. 1976), 215–226. ISSN: 03043975. DOI: 10.1016/0304-3975(76)90057-8.

[11] Jonathan F. Buss and Judy Goldsmith. "Nondeterminism within P". In: *SIAM Journal on Computing* 22.3 (1993), pp. 560–572. DOI: 10.1137/0222038.

[12] Samuel R. Buss. "The Boolean Formula Value Problem is in ALOGTIME". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. New York, New York, USA: ACM, 1987, pp. 123–131. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28409.

[13] S. Buss et al. "An Optimal Parallel Algorithm for Formula Evaluation". In: *SIAM Journal on Computing* 21.4 (1992), pp. 755–780. DOI: 10.1137/0221046.

[14] Liming Cai and Jianer Chen. "On Fixed-Parameter Tractability and Approximability of NP Optimization Problems". In: *Journal of Computer and System Sciences* 54.3 (1997), pp. 465–474. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1490.

[15] Liming Cai and Jianer Chen. "On the Amount of Nondeterminism and the Power of Verifying". In: *SIAM Journal on Computing* 26.3 (1997), pp. 733–750. DOI: 10.1137/S0097539793258295.

[16] L.M. Cai et al. "On the Structure of Parameterized Problems in NP". In: *Information and Computation* 123.1 (1995), pp. 38–49. ISSN: 0890-5401. DOI: 10.1006/inco.1995.1156.

[17] Marco Cesati. *Compendium of Parameterized Problems*. Sept. 2006. URL: http://cesati.sprg.uniroma2.it/research/compendium/.

[18] Marco Cesati and Miriam Di Ianni. "Parameterized Parallel Complexity". In: *Proceedings of the Fourth International Euro-Par Conference*. Ed. by David Pritchard and Jeff Reeve. Springer Berlin Heidelberg, 1998, pp. 892–896. ISBN: 978-3-540-49920-6. DOI: 10.1007/BFb0057945.

[19] Yijia Chen and Moritz Müller. "Bounded Variable Logic, Parameterized Logarithmic Space, and Savitch's Theorem". In: *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science*. Ed. by Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik. Budapest, Hungary: Springer, 2014, pp. 183–195. ISBN: 978-3-662-44522-8. DOI: 10.1007/978-3-662-44522-8_16.

[20] Stephen A. Cook and H. James Hoover. "A Depth-Universal Circuit". In: *SIAM Journal on Computing* 14.4 (1985), pp. 833–839. DOI: 10.1137/0214058.

[21] P. Crescenzi. "A Short Guide To Approximation Preserving Reductions". In: *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 262–273. ISBN: 0-8186-7907-7. URL: http://dl.acm.org/citation.cfm?id=791230.792302.

[22] P. Crescenzi and L. Trevisan. "On Approximation Scheme Preserving Reducibility and Its Applications". In: *Theory of Computing Systems* 33.1 (2000), pp. 1–16. ISSN: 1433-0490. DOI: 10.1007/s002249910001.

[23] Pierluigi Crescenzi and Alessandro Panconesi. "Completeness in Approximation Classes". In: *Information and Computation* 93.2 (1991), pp. 241–262. ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90025-W.

[24] Pierluigi Crescenzi et al. "Structure in approximation classes". In: *SIAM Journal on Computing* 28.5 (1999), pp. 1759–1782.

[25] P. Crescenzi et al. "Structure in Approximation Classes". In: *Computing and Combinatorics*. Ed. by Ding-Zhu Du and Ming Li. Vol. 959. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1995, pp. 539–548. ISBN: 978-3-540-60216-3. DOI: 10.1007/BFb0030875.

[26] Díaz, Josep and Torán, Jacobo. "Classes of bounded nondeterminism". In: *Mathematical Systems Theory* 23.1 (1990), pp. 21–32. ISSN: 0025-5661. DOI: 10.1007/BF02090764.

[27] J. Díaz et al. *Paradigms for fast parallel approximability*. Cambridge International Series on Parallel Computation. Cambridge University Press, 1997. ISBN: 9780521117920.

[28] Irit Dinur. "The PCP theorem by gap amplification". In: *Journal of the ACM* 54.3 (June 2007). ISSN: 0004-5411. DOI: 10.1145/1236457.1236459.

[29] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer-Verlag London, 2013. ISBN: 978-1-4471-5559-1. DOI: 10.1007/978-1-4471-5559-1.

[30] Pavlos S. Efraimidis. "The complexity of linear programming in (γ, κ)-form". In: *Information Processing Letters* 105.5 (2008), pp. 199–201. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2007.08.025.

[31] Pavlos S. Efraimidis and Paul G. Spirakis. *Fast Parallel Approximations to Positive Linear Programs with a small number of constraint violations*. Tech. rep. Computer Technology Institute, Department of Computer Engineering and Informatics, University of Patras, 1999.

[32] Michael Elberfeld, Christoph Stockhusen, and Till Tantau. "On the Space and Circuit Complexity of Parameterized Problems: Classes and Completeness". In: *Algorithmica* 71.3 (2014), pp. 661–701. ISSN: 1432-0541. DOI: 10.1007/s00453-014-9944-y.

[33] U. Feige et al. "Approximating clique is almost NP-complete". In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. IEEE. 1991, pp. 2–12. DOI: 10.1109/SFCS.1991.185341.

[34] Jörg Flum and Martin Grohe. "Describing parameterized complexity classes". In: *Information and Computation* 187.2 (2003), pp. 291–319. ISSN: 0890-5401. DOI: 10.1016/S0890-5401(03)00161-5.

[35] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-29952-3. DOI: 10.1007/3-540-29953-X.

[36] Dimitris Fotakis and Paul Spirakis. "(poly(log log n), poly(log log n))–Restricted Verifiers are Unlikely to Exist for Languages in NP". In: *Mathematical Foundations of Computer Science 1996*. Ed. by Penczek, Wojciech and Szałas, Andrzej. Vol. 1113. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996, pp. 360–371. ISBN: 978-3-540-61550-7. DOI: 10.1007/3-540-61550-4_162.

[37] *GAP — Groups, Algorithms, and Programming, Version 4.8.6*. The GAP Group. 2016. URL: http://www.gap-system.org.

[38] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. Freeman, New York, 1979. ISBN: 978-0716710455.

[39] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*. Victoria, British Columbia, Canada: ACM, 2008, pp. 113–122. ISBN: 978-1-60558-047-0. DOI: 10.1145/1374376.1374396.

[40] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-completeness Theory*. New York: Oxford University Press, 1995. ISBN: 978-0195085914.

[41] Samuel Haney. Personal communication. 2013.

[42] Johan Håstad. "Some Optimal Inapproximability Results". In: *Journal of the ACM* 48.4 (July 2001), pp. 798–859. ISSN: 0004-5411. DOI: 10.1145/502090.502098.

[43] Harry B Hunt et al. "NC-Approximation Schemes for NP- and PSPACE-Hard Problems for Geometric Graphs". In: *Journal of Algorithms* 26.2 (1998), pp. 238–274. ISSN: 0196-6774. DOI: 10.1006/jagm.1997.0903.

[44] N. Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, 1999. ISBN: 9780387986005.

[45] David S. Johnson. "A Catalog of Complexity Classes". In: *Algorithms and Complexity*. Ed. by Jan van Leeuwen. Vol. A. Handbook of Theoretical Computer Science. Elsevier, 1990. Chap. 2, pp. 68–161. ISBN: 0444880712.

[46] David S. Johnson. "Approximation Algorithms for Combinatorial Problems". In: *Journal of Computer and System Sciences* 9.3 (1974), pp. 256–278. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(74)80044-9.

[47] L. G. Khachian. “A polynomial time algorithm for linear programming”. In: *Doklady Akadmii Nauk SSSR, n.s.* 244.5 (1979), pp. 1093–1096.

[48] Sanjeev Khanna et al. “On Syntactic versus Computational Views of Approximability”. In: *SIAM Journal on Computing* 28.1 (1998), pp. 164–191. DOI: 10.1137/S0097539795286612.

[49] Chandra M. R. Kintala and Patrick C. Fischer. “Refining Nondeterminism in Relativized Polynomial-Time Bounded Computations”. In: *SIAM Journal on Computing* 9.1 (1980), pp. 46–53. DOI: 10.1137/0209003.

[50] Lefteris M. Kirousis and Paul Spirakis. “Probabilistic log-space reductions and problems probabilistically hard for P”. In: *Proceedings of the First Scandinavian Workshop on Algorithm Theory*. Ed. by Rolf Karlsson and Andrzej Lingas. Berlin, Heidelberg: Springer, 1988, pp. 163–175. ISBN: 978-3-540-39288-0. DOI: 10.1007/3-540-19487-8_19.

[51] P. Kolaitis and M. N. Thakur. “Polynomial-time optimization, parallel approximation, and fixpoint logic”. In: *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*. May 1993, pp. 31–41. DOI: 10.1109/SCT.1993.336543.

[52] Richard E. Ladner. “The Circuit Value Problem is Log Space Complete for P”. In: *SIGACT News* 7.1 (Jan. 1975), pp. 18–20. ISSN: 0163-5700. DOI: 10.1145/990518.990519.

[53] Michael Luby and Noam Nisan. “A Parallel Approximation Algorithm for Positive Linear Programming”. In: *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*. San Diego, California, USA: ACM, 1993, pp. 448–457. ISBN: 0-89791-591-7. DOI: 10.1145/167088.167211.

[54] Or Meir. “Combinatorial PCPs with efficient verifiers”. In: *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE. 2009, pp. 463–471. ISBN: 978-1-4244-5116-6. DOI: 10.1109/FOCS.2009.10.

[55] Pekka Orponen and Heikki Mannila. *On approximation preserving reductions: Complete problems and robust measures*. Tech. rep. Helsinki, Finland: Department of Computer Science, University of Helsinki, 1987.

[56] Christos H. Papadimitriou and Mihalis Yannakakis. “Optimization, approximation, and complexity classes”. In: *Journal of Computer and System Sciences* 43.3 (1991), pp. 425–440. ISSN: 0022-0000. DOI: 10.1016/0022-0000(91)90023-X.

[57] Omer Reingold, Salil Vadhan, and Avi Wigderson. “Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors”. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. IEEE. 2000, pp. 3–13. ISBN: 0-7695-0850-2. DOI: 10.1109/SFCS.2000.892006.

[58] John E. Savage. *Models of Computation: Exploring the Power of Computing*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201895390.

[59] Maria Serna and Fatos Xhafa. "On Parallel versus Sequential Approximation". In: *Proceedings of the Third Annual European Symposium on Algorithms*. Ed. by Paul Spirakis. Vol. 979. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1995, pp. 409–419. ISBN: 978-3-540-60313-9. DOI: 10.1007/3-540-60313-1_159.

[60] Maria Serna and Fatos Xhafa. "The Parallel Approximability of the False and True Gates Problems for NOR-Circuits". In: *Parallel Processing Letters* 12.1 (Mar. 2002). Ed. by Paul Spirakis, pp. 127–136. ISSN: 01296264. DOI: 10.1142/S0129626402000872.

[61] Srinath Setty et al. "Making argument systems for outsourced computation practical (sometimes)". In: *Proceedings of the 19th Annual Network and Distributed System Security Symposium*. 2012.

[62] Srinath Setty et al. "Taking Proof-Based Verified Computation a Few Steps Closer to Practicality". In: *Proceedings of the 21st USENIX Security Symposium*. Bellevue, WA: USENIX, 2012, pp. 253–268. ISBN: 978-1-931971-95-9.

[63] Till Tantau. "Logspace Optimization Problems and Their Approximability Properties". In: *Theory of Computing Systems* 41.2 (2007), pp. 327–350. DOI: 10.1007/s00224-007-2011-1.

[64] Justin Thaler et al. "Verifiable Computation with Massively Parallel Interactive Proofs". In: *Proceedings of the Fourth USENIX Workshop on Hot Topics in Cloud Computing*. Berkeley, CA: USENIX, June 2012.

[65] Luca Trevisan. "Erratum: A Correction to 'Parallel Approximation Algorithms by Positive Linear Programming'". In: *Algorithmica* 27.2 (2000), pp. 115–119. ISSN: 0178-4617. DOI: 10.1007/s004530010007.

[66] Luca Trevisan. "Parallel Approximation Algorithms by Positive Linear Programming". In: *Algorithmica* 21.1 (1998), pp. 72–88. ISSN: 0178-4617. DOI: 10.1007/PL00009209.

[67] Luca Trevisan and Fatos Xhafa. "The Parallel Complexity of Positive Linear Programming". In: *Parallel Processing Letters* 8.4 (1998), pp. 527–533. DOI: 10.1142/S0129626498000511.

[68] Marty J. Wolf. "Nondeterministic circuits, space complexity and quasigroups". In: *Theoretical Computer Science* 125.2 (1994), pp. 295–313. ISSN: 0304-3975. DOI: 10.1016/0304-3975(92)00014-I.

Curriculum vitae

Jeffrey Finkelstein (born 1988)

Boston University Computer Science Department

111 Cummington Mall

Boston, MA 02215

United States