Approximability and Completeness in the Polynomial Hierarchy

by

Christopher Matthew Umans

B.A. (Williams College) 1996

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Christos H. Papadimitriou, Chair
Professor Alistair Sinclair
Professor Dorit Hochbaum

Fall 2000

Approximability and Completeness in the Polynomial Hierarchy

by

Christopher Matthew Umans

Abstract

Approximability and Completeness in the Polynomial Hierarchy

by

Christopher Matthew Umans
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Christos H. Papadimitriou, Chair

The theoretical computer science methodology for dealing with NP optimization problems has been highly successful in exposing intractability through the theory of NP-completeness, developing approximation algorithms, and establishing limits on approximability. A number of natural and practically important problems relating to logic minimization lie beyond NP in the second level of the Polynomial Hierarchy (Σ_2^p), and have eluded attempts to apply a similar methodology for Σ_2^p. In this dissertation, we address intractability, approximability, and limits of approximability for these problems.

We prove that several of these problems are Σ_2^p-complete, including the Minimum Equivalent DNF problem, solving a well-known problem that had been open for 24 years. We present a dozen natural optimization problems that are a mix of practically important problems and more basic combinatorial problems and show that they are complete for appropriate levels of the Polynomial Hierarchy. A variant of one of these problems turns out to require only limited nondeterminism; to capture the complexity of this problem we define a new complexity class between coNP and Σ_2^p and show that the problem is complete for the new class.

We develop a technique for showing that these problems are hard to approximate to within very large factors. In contrast to the situation for NP optimization problems, our technique does not rely on Probabilistically Checkable Proofs in any way; instead we employ a new type of code, which we call an *or-code*, to give simple and powerful gap-producing reductions. We show close connections between or-codes and *dispersers*, and we give randomized and explicit constructions of or-codes based on dispersers, as well as

efficient encoding and list-decoding procedures. For all of the problems we consider, we obtain inapproximability ratios of n^ϵ, and for several, we obtain $n^{1-\epsilon}$, which is optimal up to lower order terms.

We give approximation algorithms for minimization problems in the Polynomial Hierarchy. Previously, no approximation algorithms with formal approximation guarantees were known for any of the problems we consider, despite the practical importance of several of them. Our algorithms are based on an elegant technique in learning theory, and they fit into a generic framework that can be applied to a wide array of minimization problems in the Polynomial Hierarchy. Our algorithms are actually approximation *schemes*: they exhibit a tradeoff between approximation quality and asymptotic running time. For most of the problems we consider, we achieve approximation ratios of $n/\log n$ with an expected polynomial running time.

To Yi

Contents

Acknowledgements

First and foremost, I am indebted to my advisor, Christos Papadimitriou, for his advice, support, and confidence in me from the moment I arrived at Berkeley. I would like to thank my readers, Alistair Sinclair and Dorit Hochbaum, for their comments on a draft of this thesis, and Tiziano Villa for reading and commenting on an early version of some of this material. Thanks also to my family and friends for their support, and to my fellow graduate students and the faculty, visitors, and staff at Berkeley who made my graduate school experience so enjoyable. I would also like to acknowledge the financial support of a National Science Foundation Graduate Research Fellowship, as well as support from NSF grant CCR-9626361 and NSF grant CCR-9820897.

Chapter 1

Introduction and History

Theoretical computer science strives to understand exactly how efficiently computational problems may be solved. Ideally, this understanding is achieved by exhibiting an upper bound (by designing an efficient algorithm), and establishing a lower bound (by *proving* that all algorithms with a certain efficiency are incapable of solving the problem at hand). When the upper and lower bounds match, the problem is "solved:" a theoretically optimal algorithm has been found.

While the upper bound half of this program has enjoyed steady progress and significant successes, the lower bound half is almost entirely devoid of (unconditional) results of any kind for reasonably general models of computation. In the absence of techniques for proving strong lower bounds, an extremely useful and powerful surrogate has developed, revolving around the notions of *completeness* and *complexity classes*.

Undoubtably, the most important and pervasive complexity class is NP, the class of all problems whose solutions can be efficiently verified. The vast majority of (decision) problems we might wish to attack computationally belong to NP. Of these, many important practical problems are NP-complete, meaning that every problem in NP is efficiently reducible to them. To the extent that we believe NP constitutes a genuinely more expressive class of problems than P, we also must accept that NP-complete problems are fundamentally more difficult than those in P.

The first natural problem shown to be NP-complete was 3-SAT, independently by Cook [18] and Levin [52] in the early seventies. This problem asks, given a 3-CNF Boolean formula ϕ, whether there exists an assignment that satisfies ϕ. Shortly thereafter, Karp [41] proved twenty-one natural problems NP-complete, including the first NP *optimization* problems, and within ten years hundreds of problems had been shown to NP complete.

It is widely accepted that the importance of a complexity class is measured by its abundance of natural complete problems; in this regard NP is surely the champion. Natural complete problems also serve other important functions. They stand as a concrete representative of an abstractly defined complexity class. One can prove things about the entire class by proving things about a complete problem for the class, which is often easier (for two recent examples, see [28] and [29]). They supply a "testbed" for algorithmic techniques meant to cope with the inherent difficulties of the complexity class in question. Complete problems can be used to communicate the essence of a complexity class to non-specialists or students; witness the frequency with which the Traveling Salesperson problem is mentioned in such contexts. The Traveling Salesperson problem is compelling because it is at once easy to describe and plainly relevant in many real-world applications, arguably more so than problems like 3-SAT.

Following closely on the heels of Cook, Karp and Levin, Meyer and Stockmeyer [58] built a hierarchy of complexity classes from P and NP, called the Polynomial Hierarchy (PH). They gave complete problems called QSAT (for "quantified SAT") for every level of the PH. To complexity theorists, the PH is an important part of the landscape: many theorems hold under the assumption that the PH does not collapse (see [42] and [45] for an old and new example), and a collapse is widely regarded as (almost) as unlikely as P = NP.

But a compelling and natural complete problem that was practically relevant, like Traveling Salesperson for NP, was missing, although the candidates for this role were known from the beginning. Meyer and Stockmeyer wrote in their original PH paper [58] that they were led to define the PH when considering the problem MIN FORMULA, which asks, given a Boolean formula ϕ, what is the smallest formula ϕ' equivalent to ϕ. Because (we presume) one cannot check if ϕ' is equivalent to ϕ efficiently, this problem lies beyond NP, but it also seems unlikely to be PSPACE complete. It is a famous unresolved conjecture that this problem is in fact complete for the second level of the PH, called Σ_2^p.

In a seminal 1976 journal article defining the PH, Stockmeyer [77] proposed a restriction of this problem called MIN DNF. Here, ϕ and ϕ' are required to be DNF formulae. Together with MIN FORMULA and other variants of the so-called "circuit minimization" problem, MIN DNF is often (literally) the textbook example of a natural complete problem in the second level of the PH. For over 20 years, it has been a famous open problem to prove completeness for MIN DNF. We settle this problem in work presented in this dissertation.

It is likely that the apparent difficulty in proving completeness for MIN DNF was in part due to the scarcity of natural problems known to be complete for the second level of the hierarchy. In contrast, at the NP level there is an immense collection of known complete problems, and largely as a result, the task of showing that a problem is NP-complete is by now almost routine. A reduction from a problem with similar structure is almost always easier than a reduction from, say, 3-SAT and at the NP level there are many problems to choose from. In the case of MIN DNF, the most basic identifying feature is that it is an *optimization problem*, and, prior to this work, there existed a grand total of one known Σ_2^{P}-complete optimization problem[1] [86].

One contribution of this dissertation is to correct this situation by defining more than a dozen natural Σ_2^{P} *optimization* problems, and proving them complete. In addition, our basic minimization and maximization problems have versions for every level of the PH. Having a variety of optimization problems to reduce from should make it significantly easier in the future to prove natural optimization problems complete for levels of the PH. Indeed, the most difficult step in proving MIN DNF Σ_2^{P}-complete was identifying the "correct" intermediate problem to reduce from, and showing that that problem was Σ_2^{P}-complete. The problems we define include a mix of practically relevant problems (relating to DNF minimization heuristics), and more basic combinatorial problems, in the spirit of [41].

Three decades of research on NP optimization problems points the way to the next step after proving (likely) intractability of optimization problems: investigate their approximability. Since the beginning of NP-completeness, approximation algorithms have been among the most effective ways of coping with intractability (see [35] for an in-depth account). Historically, approximation algorithms appeared long before the celebrated hardness-of-approximation results of the last decade (for example, Johnson's greedy approximation algorithm for SET COVER [38] was only proved optimal by recent technical developments in [7] and [70]). In this dissertation, we develop both approximation algorithms and inapproximability results for optimization problems in the PH. We view hardness of approximation results as a strengthening of the reductions showing completeness; hence we discuss them first.

Standard reductions between (decision versions of) optimization problems show

[1]Based on a thorough, but not infallible, literature search. The problem is a question about sets of succinctly represented integers, and is discussed in the Related Work section. Some Π_2^{P}-complete optimization problems were known [46, 54, 48, 47, 53] and a Σ_3^{P}-complete optimization problem is found in [73].

that solving the problem exactly is intractable, unless an unlikely complexity class collapse occurs. This does not rule out the possibility of approximation algorithms. To show that a problem is hard to approximate requires a *gap-producing* reduction – one that (in the minimization case) always produces for "yes" instances an instance whose optimum is less than some value k, and for "no" instances an instance whose optimum is larger than gk, for some $g > 1$. A g-approximation algorithm can distinguish between "yes" and "no" instances, and hence is as unlikely as an exact algorithm. Gap-producing reductions are more robust than ordinary reductions, and have historically been much harder to obtain.

For NP optimization problems, gap-producing reductions (for all but a very few problems) required the development of a surprising and elegant alternate characterization of NP by *probabilistically checkable proofs* (PCPs). The so-called "PCP Theorem" is attributed to Arora and Safra [6] and Arora, Lund, Motwani, Sudan, and Szegedy [5], although their work was the culmination of the contributions of many authors (see [3] for a recent survey). This characterization and the deep connection with hardness of approximation is widely regarded as one of the most prominent successes of theoretical computer science to date.

Shortly following the PCP developments and the connection to approximation, Kiwi et al. [44] built on the PCP Theorem and its techniques to characterize the levels of the PH by "Alternating Prover" proof systems. Parts of this characterization were developed independently in [46], and portions were already implicit in [17]. In a series of works [46, 54, 48, 47, 53], Ko and Lin used these results to prove that certain Π_2^p *function approximation* problems are Π_2^p-hard to approximate to within some constant factor. These problems are fundamentally different from the optimization problems we consider here: they seek to approximate the value of a function that can be computed exactly in Π_2^p; our problems are search problems in the traditional sense – we seek an optimum structure from among a set of feasible solutions, and the decision versions of our problems therefore lie in the "Sigma" levels of the PH.

In this dissertation, we establish inapproximability results for optimization problems in the PH that are optimal up to lower order terms. To put these results in perspective, it is useful to compare them to analogous results for NP optimization problems. Within NP, there are empirically four levels of inapproximability factors: constant, logarithmic, $2^{\log^{1-\delta} n}$ for any $\delta > 0$, and n^ϵ for some ϵ, although it has been conjectured [9] that the third and fourth levels collapse; i.e., that the $2^{\log^{1-\delta} n}$ ratios can be improved to n^ϵ ratios. For all of our optimization problems, we achieve inapproximability factors of n^ϵ – placing them among the "most inapproximable" problems. However, we do even better: we prove an inapproximability factor of $n^{1-\epsilon}$, for any $\epsilon > 0$, and even non-constant ϵ under stronger complexity assumptions. Such results at the NP level have been difficult to achieve; for example, a similarly strong result for CLIQUE obtained relatively recently by Håstad [32] was the culmination of a long line of research dating to the beginnings of PCPs.

Perhaps more significant than the strong inapproximability factors we achieve is the novel technique we employ to prove these results. Whereas virtually all strong inapproximability results to date (including the aforementioned results for problems in the PH) have used PCPs in a seemingly essential way, our techniques are "PCP-free" in that they do not rely on the PCP machinery or techniques in any way. They instead employ a new type of code, which we call an *or-code*, in a way that leads to simple and powerful gap-producing reductions. The quality of the reductions depends on parameters of the or-code, and we show close connections between or-codes and *dispersers*, a type of bipartite expander graph that has been the focus of intense recent research. Dispersers, first defined by Sipser [75], and a related construct called *extractors* have been used in many complexity-theoretic applications; see [61, 60] for a survey.

Expander graphs have been used before in approximation-preserving reductions *between* NP optimization problems and to amplify the inapproximability ratios for certain problems. Papadimitriou and Yannakakis [63] use sparse expander graphs to reduce MAX-3SAT to a version in which the number of occurrences of each variable is bounded by a constant. Crescenzi, Silvestri and Trevisan [21] use dense expander graphs in reductions between weighted and unweighted versions of several NP optimization problems. Zuckerman [89] uses extractors to "amplify" PCPs and show inapproximability for a variety of NP optimization problems. Our use of or-codes, based on dispersers, is fundamentally different. We use them in gap-*producing* reductions that replace the usual role of PCPs altogether; the other results require the PCP Theorem as a starting point. We use highly imbalanced dispersers – one side of the bipartite graph is polynomially large, while the other is *exponentially large*. And (as is necessary in light of the previous sentence), the graphs are represented *implicitly* in the reductions: the problem instances we produce encode, in Cook/Levin fashion, *algorithms* that compute neighbors within these graphs.

Although the or-code technique can be used to show inapproximability for a wide range of problems in the PH, it does not seem suited for proving the inapproximability of a basic optimization problem concerned with finding shortest implicants. In this dissertation we investigate the complexity and inapproximability of three variants of the shortest implicant problem and obtain some surprising results. Although all three variants appear on the surface to be Σ_2^p-complete, the DNF variant turns out to be much easier: it can be solved in quasipolynomial time with access to an NP oracle (in contrast to Σ_2^p, which probably requires exponential time with access to an NP oracle). Within NP, problems that are solvable in quasipolynomial time appear occasionally in the literature (see for example [26]), usually accompanied by a nagging uncertainty: is the quasipolynomial running time the best possible, or does a polynomial-time algorithm await discovery? In the former case (given the current limits of our lower-bound techniques), this uncertainty can be best put to rest by defining an appropriate complexity class for which the problem is complete. Papadimitriou and Yannakakis [64] succeed in classifying some problems in this way, by showing them to be complete for two new classes utilizing *limited nondeterminism*, LOGNP and LOGSNP; others in the same paper and [26] continue to elude such a classification. We succeed in classifying the DNF variant of the shortest implicant problem by showing that it is complete for the class of *all* problems that can be solved using $O(\log^2 n)$-limited nondeterminism and an NP oracle in a specific way. This shows that the problem is unlikely to be any easier than what we already know, and provides the first example of the "limited non-determinism" phenomenon in a natural problem in the PH, above NP.

For all three variants of the shortest implicant problem, we prove optimal inapproximability results up to lower order terms. For the DNF variant, the proof follows along similar lines as the inapproximability results using or-codes, although the or-codes themselves are unnecessary. For the formula variant, we obtain optimal inapproximability only after a fairly delicate construction of a family of Boolean formulae with a certain extremal property. These results continue the theme that PCPs are not necessary to show inapproximability in the PH.

We turn now to positive results (approximation algorithms) for minimization problems in the PH. Prior to this work, no approximation algorithms have appeared in the literature for PH optimization problems. We give approximation algorithms for a number of Σ_2^p minimization problems in addition to the basic minimization problems we have defined for each level of the PH. These algorithms are an extension of an elegant technique in learning theory, developed by Bshouty et al. [14], for learning Boolean circuits exactly using equivalence queries and an NP oracle. Despite the fact that we have proved that many of the problems in question are hard to approximate to within very large factors, our approximation algorithms perform provably well on "dense" instances in which the size of the instance is very large compared to the number of variables. Arora, Karger, and Karpinski [4] have observed a similar phenomenon at the NP level. In addition, our algorithms are, strictly speaking, approximation *schemes* in the sense that they all exhibit a continuous tradeoff between (asymptotic) running time and the quality of the approximation. So while in general we can only guarantee relatively poor approximation quality in polynomial time, we can guarantee a better approximation in quasipolynomial time, and so on. An appealing strategy, then, is to run the algorithm on a given instance until the desired minimization is achieved.

Having just discussed algorithms for PH optimization problems, this is perhaps an appropriate point to discuss the practical significance of our work. The MIN DNF problem (also called *two-level logic minimization*) was formulated in the 1950's by Quine [67], and since then many researchers have sought efficient heuristic and exact minimization procedures (see [19] for a detailed overview). The motivation is clear: DNF minimization is important in VLSI design, PLA (Programmible Logic Array) synthesis, and multi-level logic synthesis [23]. It also plays an important role in reliability analysis [20] and automated reasoning (for example, [71]). In addition to MIN DNF, a number of the problems we consider formalize various heuristics that are widely used in approximate and exact logic minimization [23]. Today, DNF minimization, implicant finding, and many of our other problems belong to active research areas, with much of the current focus on alternate representations by BDD's and related structures [59].

By bringing to bear the tools of computational complexity on the constellation of problems surrounding logic minimization, we aim to foster a deep understanding of their mathematical structure, and provide a solid foundation for the principled investigation of algorithmic strategies for dealing with them. Because the problems at hand lie above NP in the PH, the insights gained from this treatment are more subtle and less familiar than at the NP level, but they are significant nonetheless. NP-completeness informs us of the boundaries of what strategies are reasonable when the goal is a polynomial time solution; Σ_2^p-completeness informs of what can (or cannot) be done in polynomial time *with access to an NP oracle*. At first blush this may seem to have no practical relevance. However, almost every heuristic, approximate, or exact method for solving logic minimization problems employs a TAUTOLOGY subroutine [23] (this is necessary even to check the feasibility of a solution). If we assume the presence of such a subroutine, Σ_2^p-completeness plays *exactly* the role of NP-completeness for "ordinary" optimization problems – it distinguishes the intractable from the efficiently solvable. Similarly, non-trivial approximation algorithms that require access to an NP oracle (as ours do) are of more than theoretical interest; they are *efficient* approximation algorithms in the model appropriate for Σ_2^p optimization problems.

Stepping back, the situation is quite remarkable. The Polynomial Hierarchy is sometimes regarded as a purely syntactic construct that is technically useful, but of little use in classifying natural problems. Despite that reputation, its very invention 25 years ago was motivated by a desire to do just that: namely, to understand the complexity of logic minimization. It is clear now that it is highly successful in this regard. It seems to provide the precisely correct model for investigating the complexity of logic minimization, as empirically determined by practitioners. It is heartening to see this validation after so many years.

1.1 Related work

As noted above, logic minimization (and especially DNF minimization) has been studied since the 1950s as an important practical problem. The most famous early algorithmic results are attributed to Quine [67] and McCluskey [56]. Since then a large body of literature has developed on heuristic and exact methods for logic minimization; it is too voluminous for us to survey here. However, to our knowledge no *approximation algorithms*, in the formal sense, with provable approximation guarantees, have appeared in the literature prior to our work.

Since the definition of the Polynomial Hierarchy in the early 1970's [58, 77], the problems MIN FORMULA and MIN DNF have been famous candidates for natural Σ_2^p-complete problems. But, prior to our work, the only non-trivial hardness result for these problems was obtained only recently by Hemaspaandra and Wechsung [33]. They show that both MIN FORMULA and MIN DNF are hard for the complexity class $P_{||}^{\mathrm{NP}}$ (introduced by Papadimitriou and Zachos [65]) which lies between coNP and Σ_2^p. Their result relies on a technique developed by Wagner [85] for "boosting" certain NP-hardness results to $P_{||}^{\mathrm{NP}}$-hardness, which has been successfully applied to several natural problems (see [34]). For MIN FORMULA, the result of [33] remains the best known lower bound.

Most of the other problems we define have not been treated formally in the complexity literature, although they capture the problems being solved by many heuristics for logic minimization that are in wide use. Some problems of a similar flavor ("circuit subfunction relations") are shown by Borchert and Ranjan to be Σ_2^p-complete [12], and some related problems are studied in [13]. A result of Agrawal and Thierauf [1] identifies one of these problems, the Boolean Isomorphism Problem, as having intermediate complexity between coNP and Σ_2^p.

A number of other natural complete problems in the second level of the PH have been identified. The Σ_2^p-complete INTEGER EXPRESSION INEQUIVALENCE problem appeared in the original papers introducing the PH [58, 77]. Problems with a similar structure (deciding whether two monotonic relational expressions are equivalent, and deciding whether two context-free grammars with only one terminal letter are equivalent) were shown to be Π_2^p-complete by Sagiv and Yannakakis [72], and Huynh [36], respectively. Wagner [86] showed that several simple questions about sets of integers become Σ_2^p- and Π_2^p- complete when the input is represented succinctly (and even Σ_3^p-complete, in one case). Schaefer [74] proved two problems from graph Ramsey theory Π_2^p-complete. Ko and Tzeng [49] identified three Σ_2^p-complete problems relating to learning theory; more recently Schaefer [73] showed that computing the VC dimension, an important parameter in learning theory, is Σ_3^p-complete. This problem, together with one of the problems Wagner considers, are the only examples of *optimization* problems in the "Sigma" levels of the PH that had been shown to be complete prior to our work.

As mentioned above, Ko and Lin identified a number of *function approximation* problems that are Π_2^p-complete, and Π_2^p-hard to approximate to within some constant factor. These problems include a pattern matching problem related to program optimization [54], the problem of finding the longest circuit in a directed graph with the directions on some edges undecided [48], and a variety of min-max optimization problems [47]. We mention in passing that in a similar spirit Condon et al. [17] give some functions that are PSPACE-hard to approximate.

Finally, at the NP level, some problems relating to logic minimization have been investigated. Versions of the DNF minimization problem in which the input is given as a fully specified truth table, or the set of ones (or zeros) of the function, and some other variants are known to be NP-complete [27, 55, 66]. A recent thesis of Czort [22] is a good reference for these results. It is known that logic minimization for certain more restrictive representations than DNF is "only" NP-complete [88, 59]. Recently, Kabanets and Cai [39] observed that another NP problem, circuit minimization when the entire truth table is given as input, is unlikely to be in P, and yet proving it NP-complete would likely have interesting side-effects.

1.2 Outline

In the remainder of Chapter 1, we give some basic definitions and conventions for the remainder of the dissertation. Because MIN DNF is often described in complexity theory textbooks and courses as the prime example of a natural Σ_2^{p} problem, we provide in Chapter 2 a simple self-contained proof that MIN DNF is Σ_2^{p}-complete, suitable for presentation in these settings. Parts of this chapter appeared in [81] and will appear in a JCSS special issue for FOCS 98. In Chapter 3, we define or-codes, describe the connection with dispersers, and construct explicit or-codes with a range of parameters. In Chapter 4, we define a basic combinatorial minimization and maximization problem for each level of the PH, and use or-codes to show that these problems are inapproximable to within very large factors. In Chapter 5, we define a number of natural optimization problems in the second level of the PH, and give approximation-preserving reductions to show that they are Σ_2^{p}-complete, and hard to approximate to within very large factors. Portions of Chapters 3,4, and 5 appeared in [82]. In Chapter 6, we investigate shortest implicant problems: we define a new complexity class to capture the complexity of the DNF variant, and we give hardness of approximation results for all three variants we consider. Parts of this chapter appeared in [83]. In Chapter 7, we give a general approximation algorithm for Σ_2^{p} minimization problems, and we specialize it to give non-trivial approximation algorithms for many of the problems in this dissertation. Parts of this chapter are joint work with Osamu Watanabe. Finally, in Chapter 8 we offer some conclusions and open problems.

1.3 Preliminaries

Before we move on to more specific definitions, we gather here a few notational conventions. We will use capital letters for sets of Boolean variables, and lower case letters for individual variables in the set. The symbol $\preceq$ denotes the coordinate-wise partial order on strings in $\{0,1\}^n$. We identify 1 with TRUE and 0 with FALSE, and use them interchangeably throughout. We use $\mathrm{poly}(n)$ to mean $O(n^c)$ for some constant c.

1.3.1 The Polynomial Hierarchy

We assume the reader is familiar with standard definitions from complexity theory. We review here the definition of the Polynomial Hierarchy, and the oracle and quantifier characterizations of it:

Definition 1.1 (PH – oracle characterization) *The Polynomial Hierarchy (PH) is defined recursively by:* $\Sigma_0^P = \Pi_0^P = P$, $\Sigma_k^P = NP^{\Sigma_{k-1}^P}$ *and* $\Pi_k^P = co\text{-}\Sigma_k^P$.

Definition 1.2 (PH – quantifier characterization) *The class* Σ_k^P *is exactly those languages* L *expressible as:*

$$L = \{I : (\exists X_1)(\forall X_2)(\exists X_3)\ldots(QX_k)R(I, X_1, X_2, \ldots X_k)\},$$

where the quantifier $Q = \forall$ *if* k *is even and* $Q = \exists$ *if* k *is odd,* $|X_i| \leq \mathrm{poly}(|I|)$ *for all* i*, and* R *is a predicate in* P*.*

The class Π_k^P *is exactly those languages* L *expressible as:*

$$L = \{I : (\forall X_1)(\exists X_2)(\forall X_3)\ldots(QX_k)R(I, X_1, X_2, \ldots X_k)\},$$

where $Q = \exists$ *if* k *is even and* $Q = \forall$ *if* k *is odd,* $|X_i| \leq \mathrm{poly}(|I|)$ *for all* i*, and* R *is a predicate in* P*.*

The canonical complete problems for the levels of the polynomial hierarchy, introduced in the original PH papers, are defined below:

k-QUANTIFIED SAT (QSAT$_k$):

INSTANCE: A quantifier-free Boolean formula $\phi(X_1, X_2, \ldots X_k)$, where the X_i are disjoint sets of variables. If k is even, then ϕ is a 3-DNF formula; if k is odd then ϕ is a 3-CNF formula.

QUESTION: Is it true that $(\exists X_1)(\forall X_2)(\exists X_3)\cdots(QX_k)\phi(X_1, X_2, \ldots X_k)$? Here the quantifier Q is $\forall$ if k is even and $\exists$ if k is odd.

Theorem 1.1 QSAT$_k$ *is* Σ_k^P*-complete.*

The proof of Theorem 1.1 is in [77], and numerous textbooks on complexity theory (e.g. [62]).

1.3.2 Transformations between algorithms and circuits

We gather here a number of standard transformations that we will use heavily. These transformations turn algorithms into circuits, and circuits into 3-CNF or 3-DNF formulae; they date to the original NP-completeness results of Cook and Levin.

Lemma 1.2 *If $f : \{0,1\}^n \to \{0,1\}$ is computable in time t by a Turing Machine A, then f is computable by a circuit C of size $O(t^2)$. Moreover, given a description of A, C can be constructed in time polynomial in t and the size of the description of A.*

A classic result improves the quadratic bound on the size of the circuit in the above lemma to $O(t \log t)$ (cf. Theorem 2.8 in [84]); however, this improvement will not be necessary for our purposes. In the next lemma, we state a folklore result concerning transforming circuits into CNF or DNF form.

Lemma 1.3 *Given a circuit $C(X)$ on n variables (i.e., $|X| = n$) with size s, we can construct in polynomial time a 3-CNF (resp. 3-DNF) formula $\phi(X,Y)$ with the property that*

$$(QY)\phi(X,Y) = 1 \Leftrightarrow C(X) = 1,$$

where quantifier $Q = \exists$ (resp. $Q = \forall$). Furthermore, $|Y| \leq s$, and $|\phi| \leq O(s)$.

A consequence of Lemma 1.2, and the quantifier characterization of the levels of the PH is the following:

Lemma 1.4 *If $f : \{0,1\}^n \to \{0,1\}$ is computable in time t by an Π_k^P (resp. Σ_k^P) algorithm A, then we can build a circuit C of size $O(t^2)$, such that:*

$$f(X) = 1 \quad \Leftrightarrow \quad (\forall Y_1)(\exists Y_2)\ldots(QY_k)C(X, Y_1, Y_2, \ldots Y_k) = 1 \tag{1.1}$$

$$(resp. \quad (\exists Y_1)(\forall Y_2)\ldots(QY_k)C(X, Y_1, Y_2, \ldots Y_k) = 1), \tag{1.2}$$

where $Q = \exists$ if k is even and $Q = \forall$ if k is odd. Moreover, given a description of A, C can be constructed in time polynomial in t and the size of the description of A.

1.3.3 Definitions relating to circuit minimization

Several of the Σ_2^{P} optimization problems that we consider are related to finding *implicants* of Boolean functions. Let f be a Boolean function. An implicant of f is a conjunction C of literals that implies f; equivalently, one can think of an implicant of f as a partial assignment to f that forces its value to be 1. An *implicate* of f is a disjunction of literals implied by f; in a similar way, one can think of an implicate as a partial assignment to f that forces its value to be 0. To refer to such restrictions we use the following notation: if R is a partial assignment to the variables of f, then f_R is the function after the partial assignment has been applied.

We often think of conjunctions (and implicants in particular) as sets of literals. Using this interpretation, we can make the following elementary observation: if C_1 and C_2 are conjunctions with $C_1 \subseteq C_2$, and C_1 is an implicant of f, then C_2 is also an implicant of f. If we wish to refer to the set of *variables* in a conjunction (or disjunction) C, we use the notation $\mathrm{vars}(C)$.

An implicant C is a *prime implicant* if the only $C' \subseteq C$ that is an implicant of f is C itself. We say that a conjunction C *covers* every assignment with no variable set in contradiction to its setting in C; these assignments are *consistent* with C. An *essential prime implicant* is an implicant that covers some assignment that is covered by no other prime implicant.

1.3.4 Optimization problems, approximation, and reductions

The levels of the PH are classes of decision problems, and we will be dealing mainly with optimization problems, so we need to specify exactly what is meant by an optimization problem for one of these classes. The following definitions essentially generalize the standard definitions for NP optimization problems.

Definition 1.3 *Let $\mathcal{C}$ be a class of decision problems. Define $\exists\mathcal{C}$ to be the class of languages L expressible as:*

$$L = \{X : \exists Y \text{ such that } R(X,Y)\},$$

where $|Y| \leq \mathrm{poly}(|X|)$, and R is a predicate in $\mathcal{C}$.

Notice that $\Sigma_k^{\mathrm{P}} = \exists\Pi_{k-1}^{\mathrm{P}}$ in this terminology. We can now define optimization problems for any such class:

Definition 1.4 ($\exists\mathcal{C}O$ problems) *Let $\mathcal{C}$ be a class of decision problems. An $\exists\mathcal{C}$ optimization problem ($\exists\mathcal{C}O$ problem) A is a 4-tuple $(I, \mathrm{sol}, m, type)$, where:*

- *I is a set of* instances[2] *of A.*
- *For every instance $x \in I$, $\mathrm{sol}(x)$ is a set of* feasible solutions *to x. The solutions are short (i.e., for all $y \in \mathrm{sol}(x)$, $|y| \leq \mathrm{poly}(|x|)$) and they are recognizable by a predicate in $\mathcal{C}$.*
- *For every instance $x \in I$ and solution $y \in \mathrm{sol}(x)$, the* measure *of y is a positive integer $m(x, y)$. The function $m(x, y)$ is computable in polynomial time.*
- $\mathrm{type} \in \{\max, \min\}$.

The goal of an $\exists\mathcal{C}O$ problem $A = (I, \mathrm{sol}, m, \mathrm{type})$, given an instance $x \in I$, is to find a optimum feasible solution y; i.e., a feasible solution $y \in \mathrm{sol}(x)$, for which $m(x, y) = \mathrm{type}\{m(x, y') : y' \in \mathrm{sol}(x)\}$. The measure of such a solution is denoted $\mathrm{opt}(x)$. The next definition is standard:

Definition 1.5 (performance ratio) *Given an instance x and a feasible solution $y \in \mathrm{sol}(x)$, the* performance ratio *of y with respect to x is:*

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

Definition 1.6 (approximation algorithm) *Let $A = (I, \mathrm{sol}, m, type)$ be an $\exists\mathcal{C}O$ problem, and let T be an algorithm with oracle access to $\mathrm{sol}(x)$, that on input x, returns a feasible solution $y \in \mathrm{sol}(x)$. For an arbitrary function $r : \mathbb{N} \to [1, \infty)$, we say that T is an* $r(n)$-approximate algorithm for A *if for all $x \in I$:*

$$R(x, T(x)) \leq r(|x|).$$

As usual, we measure the approximation ratio in terms of $|x|$, the bit-length of the string x but in certain settings a different measure of size will be more appropriate, and we will clearly state when such an alternative measure is being used.

The decision version of an $\exists\mathcal{C}O$ minimization problem $A = (I, \mathrm{sol}, m, \min)$ is: "Given an instance $x \in I$, and an integer k, is there a feasible solution $y \in \mathrm{sol}(x)$ for which

[2] It is usually also required that I be recognizable in polynomial time. We will be considering "promise" problems for which this will not necessarily be true.

$m(y) \leq k$?" This problem is in the class $\exists\mathcal{C}$. To show the optimization problem is $\exists\mathcal{C}$-hard to approximate within a factor g, we give a reduction from a known $\exists\mathcal{C}$-complete problem to a *gap-decision version* of A. We define this gap-decision version below:

Definition 1.7 (gap-decision version) *Given an $\exists\mathcal{C}O$ problem $A = (I, \mathrm{sol}, m, \min)$ (resp. $A = (I, \mathrm{sol}, m, \max)$), the $\exists\mathcal{C}$ decision problem A WITH GAP $g(n)$ is the promise problem whose instances are pairs (x, k) where $x \in I$ and k is a positive integer. The YES and NO instances are:*

- *I_{yes} is the set of instances (x, k) for which $opt(x) \leq k$ (resp. $opt(x) \geq k$).*
- *I_{no} is the set of instances (x, k) for which $opt(x) > g(|x|) \cdot k$ (resp. $opt(x) < k/g(|x|)$).*

The gap-decision problem A WITH GAP $g(n)$ captures the difficulty of approximating A within a factor $g(n)$ in the sense that a $g(n)$-approximation algorithm for A can distinguish between YES and NO instances.

A reduction from the problem A WITH GAP g to the problem B WITH GAP g' is called a *gap-preserving reduction*. This is nothing more than an ordinary reduction between two gap-decision problems; i.e., the reduction is efficient (runs in polynomial time), and it maps "yes" instances of A WITH GAP g to "yes" instances of B WITH GAP g' and "no" instances to "no" instances. In this setting, g and g' may be constants, or they may be functions of some measure of the size of the instance. In the latter case, we keep track of how reductions affect the various size measures when stating them. Note that gap-preserving reductions do not automatically give a method of transforming an approximation algorithm for B into an approximation algorithm for A; however, in all of our reductions an efficient algorithm mapping solutions to solutions will be implicit in the proofs.

Chapter 2

The Basic Reduction

In this chapter we present a simple and self-contained proof that MIN DNF is Σ_2^p-complete. We give the reduction in two steps. The argument is straightforward when the "correct" intermediate problem is used; for this purpose, we define the SHORTEST IMPLICANT CORE problem below.

2.1 From QSAT to Shortest Implicant Core

We first reduce QSAT_2 to the following problem, which shows, intuitively that shortening even a single term of a DNF optimally is hard.

SHORTEST IMPLICANT CORE (DECISION VERSION):

INSTANCE: A DNF formula ϕ, an implicant C of ϕ, and an integer k.

QUESTION: An implicant $C' \subseteq C$ of ϕ of length at most k.

Theorem 2.1 SHORTEST IMPLICANT CORE (DECISION VERSION) *is Σ_2^p-complete.*

Proof: Given an instance of QSAT_2, $\phi(x, y)$, with $|x| = n$, let S be the set of $2n$ literals $\{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}\}$. Consider the coNP algorithm A, which expects as its input a subset $S' \subseteq S$ represented as a $2n$-bit vector, with a 1 in position i if and only if element i is in S'. Algorithm A accepts an input $S' \subseteq S$ if $|S'| \geq n + 1$, or if S' contains exactly one literal for each i, and the implied assignment X to the x variables satisfies the QSAT_2 instance (i.e., $\forall y\ \phi(X, y) = 1$); otherwise it rejects.

The set of strings accepted by A is non-empty and monotone; that is, if S' is accepted by A, then all $S'' \supseteq S'$ are accepted by A. It is easy to verify that A accepts a string with weight n if and only if the instance of QSAT_2 is a positive instance.

By Lemma 1.4 and Lemma 1.3, we can construct from A a polynomially large 3-DNF $\phi'(x, y)$ with $|x| = 2n$ and with the property that $\forall y\ \phi'(x, y) \Leftrightarrow A$ accepts input x. We take our instance of SHORTEST IMPLICANT CORE to be $\langle \phi', C = x_1 x_2 \dots x_{2n}, n \rangle$. Notice that C is an implicant of ϕ' because A accepts a non-empty monotone set, so in particular it accepts the all-ones input z, which implies that $\forall y\ \phi'(z, y)$.

If the instance of QSAT_2 is a positive instance, then there is some string of weight n accepted by A, and by monotonicity, the conjunction of the variables corresponding to the ones in that string is an implicant of ϕ' contained in C; hence the instance of SHORTEST IMPLICANT CORE is a positive instance.

Conversely, if the instance of SHORTEST IMPLICANT CORE is a positive instance, then there is an implicant $C' \subseteq C$ of ϕ' of size n. Therefore, if z is an assignment to the x variables with x_i set to 1 if and only if $x_i \in C'$, then we have $\forall y\ \phi'(z, y)$, since C' is an implicant. This implies that A accepts a string with weight n, which implies that the QSAT_2 instance is a positive instance. ■

2.2 From Shortest Implicant Core to Min DNF

From here, we reduce to the decision version of MIN DNF. We first recall the definition of the problem:

MIN DNF (DECISION VERSION):

INSTANCE: A DNF formula ϕ and an integer k.

QUESTION: A DNF formula ϕ' with at most k occurrences of literals, and for which $\phi' \equiv \phi$.

Theorem 2.2 MIN DNF (DECISION VERSION) *is Σ_2^p-complete.*

Proof: We reduce from SHORTEST IMPLICANT CORE to MIN DNF. The general idea of the reduction is to construct a DNF in which all but one term *must* occur in any equivalent DNF. The size of the minimum DNF is then determined by the size of the smallest implicant core contained in that term. Given an instance of SHORTEST IMPLICANT CORE, $\langle \phi, C, k \rangle$, we perform the reduction in two steps. First, we produce from ϕ another DNF formula $\phi' \vee C$ which is equivalent to ϕ and has the additional property that every assignment covered by C falsifies every term of ϕ'. Next, we modify this formula to produce a DNF formula ϕ'' for which (in essence) all terms except the last term must be part of a minimum equivalent DNF. The size of the minimum equivalent DNF for ϕ'' is then directly related to the size of the shortest implicant core of the original instance.

To begin, let $\phi' = \phi \wedge \neg C$. When ϕ' is multiplied out into DNF, its length remains polynomial in the length of ϕ. Let $s_1, s_2, \ldots s_m$ be the terms of ϕ' when it is multiplied out into DNF, and let $m' = \max(m, k+1)$. We define the following formula:

$$\begin{aligned} \phi'' &= t_n w_1 w_2 \ldots w_{m'} \vee \bigvee_{i=1}^{m} s'_i, \\ & \quad \text{where } s'_i = s_i w_1 w_2 \ldots w_{i-1}\, w_{i+1} \ldots w_{m'}. \end{aligned}$$

Let $k' = k + m' + \sum_{i=1}^{m} |s'_i|$. Our intended instance of MIN DNF is $\langle \phi'', k' \rangle$. We begin with the easy half of the proof:

Claim 2.2.1 *If C' is an implicant of ϕ for which $C' \subseteq C$, then*

$$\phi'' \equiv C' w_1 w_2 \ldots w_{m'} \vee \bigvee_{i=1}^{m} s'_i.$$

Proof: Removing literals from a term of ϕ'' does not alter the function computed, as long as the modified term is an implicant of ϕ''. Therefore it is sufficient to show that $C' w_1 w_2 \ldots w_{m'}$ is an implicant of ϕ''. Notice that $\phi \equiv \phi' \vee C \equiv \phi''_B$, where $B = \{w_1, w_2, \ldots w_{m'}\}$. Since C is an implicant of ϕ, it is also an implicant of ϕ''_B, and therefore $C' w_1 w_2 \ldots w_{m'}$ is an implicant of ϕ''. ∎

If the implicant C in the previous claim has length $\leq k$, then the equivalent DNF given in the claim has length $\leq k'$; hence the instance of MIN DNF is a positive instance if the instance of SHORTEST IMPLICANT CORE is a positive instance.

Now, let σ be a DNF equivalent to ϕ'' of length at most k'. We show that the instance of SHORTEST IMPLICANT CORE must be a positive instance. We begin by showing that σ must include the term s'_i (in a loose sense), for all i.

Claim 2.2.2 *(1) For all i, σ includes a term s''_i such that $s'_i \subseteq s''_i$, (2) the collective length of these terms is at least $\sum_{i=1}^{m} |s'_i|$, and (3) none of these terms covers any assignment covered by $t_n w_1 w_2 \ldots w_{m'}$.*

Proof: Let A_i be an assignment consistent with s'_i and with w_i mapped to 0, and let s''_i be a term of σ that covers A_i. If s''_i excludes any w_j for $j \neq i$, then s''_i covers an assignment in which w_j and w_i are mapped to 0, but ϕ'' covers no such assignments. If s''_i excludes any literal from s_i, then s''_i covers an assignment inconsistent with s'_i, but with w_i mapped to 0. This is a contradiction, since the only assignments with w_i mapped to 0 that make ϕ'' true are consistent with s'_i. So $s'_i \subseteq s''_i$, which implies that $|s'_i| \leq |s''_i|$. Since s''_i contains the literal w_j for all $j \neq i$, it cannot cover A_j for $j \neq i$. Since the A_i are all distinct, the s''_i are distinct, so their collective length is at least $\sum_{i=1}^{m} |s''_i| \geq \sum_{i=1}^{m} |s'_i|$. Finally, no s''_i covers any assignment covered by $Cw_1w_2 \ldots w_{m'}$, since $s'_i \subseteq s''_i$ and our initial modification to ϕ ensures that s'_i and $Cw_1w_2 \ldots w_{m'}$ are disjoint. ■

We now continue with the main proof. In addition to the s''_i terms, σ must include terms that collectively cover all the assignments covered by $Cw_1w_2 \ldots w_{m'}$. Since $|\sigma| \leq k'$, these additional terms must have total length at most $k' - \sum_{i=1}^{m} |s'_i| \leq k + m'$. Any implicant of ϕ'' that covers an assignment covered by $Cw_1w_2 \ldots w_{m'}$ must include the literals $w_1w_2 \ldots w_{m'}$; otherwise it would cover an assignment consistent with C but with some w_i set to false, and ϕ'' covers no such assignments (recall our initial modification to ϕ). Therefore, each of the additional terms has length at least m', and since $k + m' < 2m'$, there must be only one such additional term. This term, C', has length at most $k + m'$, and $C' \subseteq Cw_1w_2 \ldots w_{m'}$. As noted above, $\phi''_B \equiv \phi$, where $B = \{w_1, w_2, \cdots, w_{m'}\}$. Hence, letting $C'' = C' \setminus B$, we have $1 \equiv \phi''_{C'} \equiv (\phi''_{C''})_B \equiv (\phi''_B)_{C''} \equiv \phi_{C''}$, so C'' is an implicant of ϕ, with $C'' \subseteq C$ and $|C''| \leq k$; in other words, the instance of SHORTEST IMPLICANT CORE is a positive instance. This completes the reduction from SHORTEST IMPLICANT CORE to MIN DNF. ■

We note that this proof also shows that a seemingly "easier" DNF minimization problem, which requires only "term-wise" minimality, is also Σ_2^p-complete. In this problem, given $\phi = t_1 \vee t_2 \vee \cdots \vee t_n$ and a bound k, we are only asked for an equivalent DNF ϕ' with at most k occurrences of literals of the following form: $\phi' = t'_1 \vee t'_2 \vee \cdots \vee t'_n$, and for all i, $t'_i \subseteq t_i$.

Chapter 3

Or-Codes

In this chapter we introduce a new type of code: the *or-code*. These codes are the primary technical tools used in the inapproximability results in Chapter 4. We give randomized and explicit constructions of good or-codes for a wide range of parameters, as well as efficient encoding and list-decoding procedures.

3.1 Or-Codes: definition and motivation

There are two fundamental characteristics common to many of the optimization problems we consider in this work. The first is that the measure of optimality is the *size* of the structure sought (the existentially quantified object in the quantifier characterization of Σ_2^{P}). The second is *monotonicity*: any augmentation of a feasible solution is also a feasible solution. (A similar feature of a class of NP optimization problems is examined by Kolaitis and Thakur [51]). Within NP, these characteristics are exemplified by SET COVER; within Σ_2^{P}, SUCCINCT SET COVER (defined in Section 5.3) is a prime example.

Or-codes provide a means of encoding strings (say, satisfying assignments) that is well-suited for use in reductions involving minimization problems with these characteristics. The (Hamming) weight of each codeword is small, and they are robust to significant "augmentation" – 0's getting flipped to 1's. We use them in reductions to encode proofs of membership in such a way that a valid proof has a low weight encoding, and all but the largest weight received words can be decoded, thereby forcing an acceptable received word to have very large weight if there is no valid proof.

To explain the name "or-code," we first propose the following alternative way of viewing ordinary binary codes. Using such a code, which we might term an "xor-code", we can encode an n-bit message x in a longer bitstring $C(x)$ of length M that is robust against errors that XOR an error bitstring E of length M with $C(x)$. That is, given a received word $R = C(x) \oplus E$, we can recover x, or a short list containing x, from R. For such codes, we can hope to tolerate an error rate approaching $1/2$, or in the terminology of this paragraph, an error bitstring with weight approaching $1/2M$.

Using an or-code, we can encode an n-bit message x in a longer bitstring $C(x)$ of length M as before. Here, the encoded string is robust against errors that OR an error bitstring E of length M with $C(x)$. In other words, given a received word $R = C(x) \vee E$, we can recover x, or a short list containing x, from R. In contrast to xor-codes, with or-codes we can hope to tolerate an error rate approaching 1 – an error bitstring with weight approaching M.

A second difference between xor-codes and or-codes that is critical for our intended application is the following: or-codes can have a very small maximum weight while tolerating error bitstrings of weight $\Omega(M)$ (the weight can be independent of M), while the maximum weight of xor-codes must be larger than the weight of the error bitstrings it can tolerate (otherwise, every codeword would appear in the decoding list when the received word was the all-zero string).

We now formally define or-codes:

Definition 3.1 (or-code) *An* (n, M, K, ϵ) or-code with weight W *is a subset* $C \subseteq \{0,1\}^M$ *of size* 2^n*, in which every* $y \in C$ *has weight* W *and which has the following property: for every received word* R *of weight at most* $(1-\epsilon)M$*, there are at most* K *distinct strings* $y \in C$ *for which* $y \preceq R$*.*

An (n, M, K, ϵ) *or-code* C *with weight* W *is* explicit *if the following hold:*

- *There is an injective encoding function* **Enc** : $\{0,1\}^n \to \{0,1\}^M$ *whose image is* C*, and* **Enc** *is computable in time* $\mathrm{poly}(\max(K, M))$.
- *There is a list-decoding function* **Dec** *that maps* $\{0,1\}^M$ *to a subset of* $\{0,1\}^n \cup \{FAIL\}$ *of size at most* K*. For* $R \in \{0,1\}^M$ *with weight greater than* $(1-\epsilon)M$*,* $\mathbf{Dec}(R) = \{FAIL\}$*; otherwise,* $\mathbf{Dec}(R) = \{x \in \{0,1\}^n : \mathbf{Enc}(x) \preceq R\}$[1]. **Dec** *should be computable in time* $\mathrm{poly}(\max(M, K))$.

[1] A weaker requirement that $\mathbf{Dec}(R)$ include every x for which $\mathbf{Enc}(x) \preceq R$ (and possibly other x in addition) would also suffice; using **Enc**, we could eliminate those x in $\mathbf{Dec}(R)$ for which $\mathbf{Enc}(x) \npreceq R$.

3.2 Or-Codes and dispersers

Or-codes are very similar to another combinatorial object that has been the focus of much research in recent years, namely, a *disperser.* In this section we define dispersers, comment on their relationship to or-codes, and then describe how to obtain explicit or-codes from dispersers.

We first introduce notation used to represent neighbors in a graph. Given a graph $G = (V, E)$, we denote by $\Gamma(v)$ the set of all neighbors of v in G; if V' is a subset of the vertices, then $\Gamma(V') = \cup_{v \in V'} \Gamma(v)$.

Definition 3.2 (Disperser) *An* (n, m, k, ϵ) disperser with degree 2^d *is a bipartite graph* $G = (U = [2^n], V = [2^m], E)$ *with left-degree* 2^d *and with the following property: for every subset* $U' \subseteq U$ *of size at least* 2^k, $|\Gamma(U')| > (1-\epsilon)|V|$. *We sometimes view a disperser as a function* $G : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$. *A disperser is* explicit *if the function* $G(u, i)$ *that computes the ith neighbor of vertex* $u \in U$ *can be computed in time polynomial in* n.

Ignoring for the moment the explicitness requirements of or-codes and dispersers, we see that the combinatorial objects are quite similar. In particular, given a (n, m, k, ϵ) disperser with degree 2^d, it is easy to see that $\{\Gamma(x) | x \in \{0,1\}^n\}$ is a $(n, 2^m, 2^k, \epsilon)$ or-code with weight 2^d. However, we observe that this natural correspondence works only in one direction; an (n, M, K, ϵ) or-code C with weight W does not require that the OR of K strings in C have weight $> (1-\epsilon)M$, as would be necessary for it to be a disperser in disguise.

The above construction of an or-code from a disperser also provides an efficient encoding function, if the disperser is explicit. To allow efficient list-decoding, we need to work harder. The next theorem shows how to construct explicit or-codes from explicit dispersers; we will use it heavily:

Theorem 3.1 *Given an explicit* $(n+1, m, k, \epsilon)$ *disperser with degree* $D = 2^d$, *we can construct an explicit* $(n, M = 2^m, K = 2^k, \epsilon)$ *or-code with weight* $n2^d$.

Proof: Let $G = (U, V, E)$ be the bipartite graph associated with the disperser. We think of (all but one of) the vertices in U as being organized into a complete binary tree with 2^n leaves and we identify these leaves with the set of messages $\{0,1\}^n$. For each $x \in \{0,1\}^n$

define the set $V_x \subseteq V$ to be $\Gamma(P_x)$, where P_x is the set of vertices along the unique path from the root to leaf x.

The encoding of x is simply the characteristic vector of the subset $V_x \subseteq V$. As G is an *explicit* disperser, this can be computed in $D\text{poly}(n) \leq \text{poly}(\max(K, M))$ time, which satisfies the requirement for the encoding function to be efficiently computable.

Decoding is only slightly more complicated. Let $R \in \{0,1\}^M$ be a received word, and let $V' \subseteq V$ be the associated subset. If $|V'| > (1-\epsilon)M$, then the decoder returns $\{\text{FAIL}\}$ as required.

Otherwise, we proceed with list-decoding. Say that a vertex $u \in U$ is *active with respect to* V' if $\Gamma(u) \subseteq V'$. By the disperser property of G, there are at most $K-1$ active vertices with respect to any subset V' of size at most $(1-\epsilon)M$ (because the set of active vertices fails to disperse). We need to be able to identify the set of all V_x for which $V_x \subseteq V'$. This is accomplished with the decoding procedure described below, which performs a breadth-first-search from the root of the tree, pruning the search whenever a non-active vertex is encountered:

```
Decode(V')
  Precondition: V' ⊆ V and |V'| ≤ (1 − ε)M
  Returns: {x : V_x ⊆ V'}
  if Γ(root) ⊈ V' then return {}
  else active ← root
  for i ← 1 to n do
      active ← {u ∈ U : u is a child of some u' ∈ active
                          and Γ(u) ⊆ V'}
  return active
```

The total running time of this procedure is

$$O((\text{time to compute } \Gamma(u)) \times (\#\text{ active vertices})) = \text{poly}(n)DK.$$

The input has size M; therefore, the total running time is $\text{poly}(\max(M, K))$ as required. ■

Remark 3.1 *Trevisan's disperser [80] and related dispersers [69] admit an even simpler encoding: each x is encoded as the characteristic vector of the subset $\Gamma(x) \subseteq V$. These dispersers have the property that given a subset $V' \subseteq V$ of size at most $(1-\epsilon)M$,* every x

for which $\Gamma(x) \subseteq V'$ (x viewed as a function on $\log n$ bits) can be computed with a circuit of size polylog(M) that has oracle access to V'. Decoding can be accomplished by enumerating all such circuits. A two phase decoding process is even more efficient: every x for which $\Gamma(x) \subseteq V'$ has a circuit of size only $O(\log M)$ on $O(\log n)$ bits with oracle access to V' that computes a string that is close Ecc(x), where Ecc($\cdot$) is a binary error-correcting code; the efficient list-decoding of [31] can then be used to recover x.

3.3 Efficient probabilistic construction of dispersers

Currently known explicit disperser constructions are not yet optimal for our purposes; however, simple probabilistic constructions of optimal dispersers and extractors are well known [68], and these lead to improved or-codes. For dispersers, the probabilistic constructions amount to choosing D (the degree) neighbors for each $x \in \{0,1\}^n$ independently and uniformly at random.

We use or-codes, constructed from dispersers, in a reduction, and we would like to be able to use optimal dispersers, constructed probabilistically, in a *randomized reduction.* However, the current constructions require far too many random bits for this purpose (exponentially many).

In this section we observe that for (n, m, k, ϵ) dispersers with degree d, 2^{k+d}-wise independence suffices to construct optimal dispersers; hence, if k and d are logarithmic in n, we get an efficient randomized construction. The analysis follows [68] closely.

Theorem 3.2 *Fix N, $N > K > 1$, $M > 0$, and $1 \geq \epsilon > 1/M$. Suppose*

$$D \geq 2 \cdot \max\left\{\frac{1}{\epsilon}\left(\ln\left(\frac{N}{K}\right)+2\right), \frac{M}{K}\left(\ln\left(\frac{1}{\epsilon}\right)+2\right)\right\}.$$

Let $\mathcal{H} = \{h_i : [ND] \rightarrow [M]\}$ be a KD-wise independent hash family[2]. If we define $\text{DISP}_i : [N] \times [D] \rightarrow [M]$ as:

$$\text{DISP}_i(x, y) \stackrel{def}{=} h_i(xy).$$

then $\Pr_i[\text{DISP}_i$ is not a $(n, \log M, \log K, \epsilon)$ disperser $] \leq e^{-K}$.

[2]For concreteness, we can use the following well-known $t = KD$-wise independent hash family that hashes $n = \log N + \log D$ bits to $m = \log M$ bits: a hash function in this family is a degree $t-1$ polynomial p over a finite field F of size $2^{\max n,m}$, specified by its t coefficients. The output of the hash function p on input x is the first m bits of $p(x)$. Functions in this hash family have description size poly(t, n, m) and can be evaluated in time poly(t, n, m).

Proof: Let $L = \lceil \epsilon M \rceil$. Notice that DISP_i is not a $(n, \log M, \log K, \epsilon)$ disperser if and only if there exists a subset $U \subseteq [N]$ of size K and a subset $V \subseteq [M]$ of size L for which $\Gamma(U) \cap V = \emptyset$. For every pair of subsets U and V as described, we have:

$$\begin{aligned}
\Pr_i[\Gamma(U) \cap V = \emptyset] &= \Pr_i\left[\bigwedge_{x \in U, y \in [D]} h_i(xy) \notin V\right] \leq (1 - L/M)^{KD} && (3.1)\\
&\leq \exp\left(-\frac{LKD}{M}\right) && (3.2)
\end{aligned}$$

where (3.1) is true because h_i is chosen from a KD-wise independent hash family. By our choice of D, we have the following inequalities:

$$\begin{aligned}
D &\geq \frac{2}{\epsilon}(2 + \ln(N/K)) \geq \frac{2M}{L}(2 + \ln(N/K)) \\
\exp\left(\frac{DL}{2M}\right) &\geq \exp(2 + \ln(N/K)) = e^2(N/K) && (3.3)\\
D &\geq \frac{2M}{K}(2 + \ln(\frac{1}{\epsilon})) \geq \frac{2M}{K}(2 + \ln(M/L)) \\
\exp\left(\frac{DK}{2M}\right) &\geq \exp(2 + \ln(M/L)) = e^2(M/L) && (3.4)
\end{aligned}$$

Therefore, using (3.3) and (3.4), we have:

$$\begin{aligned}
\binom{N}{K}\binom{M}{L} &\leq (eN/K)^K (eM/L)^L \\
&\leq \exp\left(\frac{DL}{2M} - 1\right)^K \exp\left(\frac{DK}{2M} - 1\right)^L = \exp\left(\frac{LKD}{M} - K - L\right).
\end{aligned}$$

There are $\binom{N}{K}$ ways to pick the set U and $\binom{M}{L}$ ways to pick the set V; the bound in (3.2) holds for any U and V, so by the union bound:

$$\begin{aligned}
\Pr_i[\text{DISP}_i \text{ is not a } (n, \log M, \log K, \epsilon) \text{ disperser }] &\leq \binom{N}{K}\binom{M}{L} \exp\left(-\frac{LKD}{M}\right) \\
&\leq \exp(-K - L) \leq e^{-K}.
\end{aligned}$$

■

We note that if K and D are polynomial in n, this construction yields an explicit disperser; however, for larger K and D, the disperser runs in time $\text{poly}(K, D)$, which is sufficient when it is used in conjunction with Theorem 3.1.

#	weight W	list size K	construction time	reference
1	$n(n+1)^c\left(\frac{M}{\epsilon^2}\right)^{4/5}$	$(n+1)^{-c}\left(\frac{M}{\epsilon^2}\right)^{1/5}$	poly(n)	Thm. 3.3
2	$n\left(\frac{n^{\log n}}{\epsilon}\right)^{c/\log\alpha}$	M^α	poly(n)	Thm. 3.4
3	$c\left(\frac{n^2}{\epsilon}\right)$	M	poly(M, n, ϵ^{-1}) (randomized)	Thm. 3.2
4			poly(M, n, ϵ^{-1}) (expected, with NP oracle)	Thm. 3.2
5			poly(M, n, ϵ^{-1})) (with Σ_2^{P} oracle)	Thm. 3.2

Table 3.1: Or code constructions for Theorem 3.5.

3.4 Constructions of explicit or-codes

Using known explicit disperser constructions and the probabilistic construction of the previous section, together with Theorem 3.1, we construct explicit or-codes with a range of parameters.

The two explicit disperser constructions that we will use are due to Srinivasan and Zuckerman [76], and Raz, Reingold, and Vadhan [69]; the relevant parameters of the two constructions are in the the following two theorems:

Theorem 3.3 ([76, Lemma 3.2]) *For every n, $k < n$, and $\epsilon > 0$, there exists an explicit $(n, m = k + d - 2\lceil\log(\frac{1}{\epsilon})\rceil - 2, k, \epsilon)$ disperser with degree $d = 4k + O(\log n)$.*

Theorem 3.4 ([69, Theorem 2]) *For every n, $k < n$, $\epsilon > 0$, and $\alpha > 0$, there exists an explicit $(n, m = (1-\alpha)k, k, \epsilon)$ disperser with degree $d = O(\log^2 n \log(\frac{1}{\epsilon})/\log(\frac{1}{1-\alpha}))$.*

Putting everything together, we obtain or-codes with parameters described in Table 3.1. In general, there is a tradeoff between the complexity of construction and the quality of the or-code parameters; we can hope for a weight W that is polynomial is n and $1/\epsilon$, and we can hope for the list size K to be bounded by M (even $\approx M/W$ is possible, although this is not necessary for our purposes).

Theorem 3.5 (or-codes) *For any n, M, and $\epsilon > 1/M$, the explicit or-codes in Table 3.1 exist. In the table the value c is an unspecified fixed constant, while $\alpha > 1$ is arbitrary. The "construction time" entry refers to the complexity of an initial construction phase to obtain a description of the or-code with the desired parameters; after this computation, the* **Enc** *and* **Dec** *functions are available and behave as specified in Definition 3.1.*

Proof: (Of 1.) Let c' be the constant hidden in the $O(\log n)$ in the statement of Theorem 3.3. Pick $k = \frac{1}{5}(\log M - c' \log(n+1) + 2\lceil \log(\frac{1}{\epsilon}) \rceil + 2)$. Theorem 3.3 gives us a $(n+1, \log M, k, \epsilon)$ disperser, with $d = 4k + c' \log(n+1)$. By Theorem 3.1, we can use this disperser to build an $(n, M, 2^k, \epsilon)$ or-code, with weight

$$W = n2^d = n(n+1)^{c'} \left(\frac{4M}{(n+1)^{c'} \epsilon^2} \right)^{4/5} = n(n+1)^{c'/5} \left(\frac{M}{\epsilon^2} \right)^{4/5},$$

and we take $c = c'/5$ to get the W stated in the Theorem.

(Of 2.) Let $\alpha' = (\alpha - 1)/\alpha$, and let c be the constant hidden in the big-oh bound on d in the statement of Theorem 3.4. Pick $k = \alpha \log M$. Theorem 3.4 gives us a $(n+1, (1-\alpha')k = \log M, k, \epsilon)$ disperser with $d = c(\log^2 n \log(\frac{1}{\epsilon}) / \log \alpha)$. By Theorem 3.1, we use this disperser to build an $(n, M, 2^k, \epsilon)$ or-code, with weight

$$W = n2^d = n \left(\frac{n^{\log n}}{\epsilon} \right)^{c/\log \alpha}.$$

(Of 3.) Set $K = M$, and $D = 2(\ln 2^{n+1} + 2)/\epsilon$. Using $O(KD \max(n + \log D, \log M))$ bits, we choose a random KD-wise independent hash function (see Appendix). By Theorem 3.2, this defines a $(n+1, \log M, \log K, \epsilon)$ disperser with probability at least $1 - e^{-K}$. By Theorem 3.1, we can use this disperser to build an (n, M, K, ϵ) or-code, with weight $nD = O(n^2/\epsilon)$.

(Of 4.) We use the same construction as above, based on Theorem 3.2. We observe that given an explicit (size $O(KD \max(n + \log D, \log M))$) description DISP of a candidate $(n+1, \log M, \log K, \epsilon)$ disperser with left-degree D, the question "Is DISP a $(n+1, \log M, \log K, \epsilon)$ disperser?" is answerable with a single co-NP query. Therefore, with access to an NP oracle, we can recognize when the probabilistic disperser construction fails, and try again.

(Of 5.) Again using the same construction as 3, we can use a Σ_2^{P} oracle to find a choice of random bits that yields an $(n+1, \log M, \log K, \epsilon)$ disperser. ■

Chapter 4

Hard Optimization Problems for the Polynomial Hierarchy

In this chapter we define a class of optimization problems that will be the building blocks for later reductions and hardness of approximation results. The minimization versions of these problems we call the *monotone minimum weight word* problems; there is a version for each level in the PH. These problems may be thought of as generalizations of the NP optimization problem MONOTONE MINIMUM SATISFYING ASSIGNMENT, which has been examined in [30] and [25]. The maximization version gives problems for each level of the PH, which we call *monotone maximum zeros* problems. Both problems rely on or-codes to show that they are hard to approximate; interestingly, the critical parameter of the or-codes used in the reduction is different in the minimization and maximization cases. The minimization problems require small weight W relative to M, while the maximization problems require very high error tolerance (i.e., very small ϵ).

4.1 A non-deterministic circuit model

An instance of a monotone minimum weight word problem (and a monotone maximum zeros problem) is a non-deterministic circuit with a certain acceptance criterion; we define such circuits first:

Definition 4.1 *A* non-deterministic (ND) circuit with k alternations *is a directed acyclic graph with arbitrary fan-out and fan-in at most two. There is a single sink node, the* **output**

gate, fan-in one **not** *gates, fan-in two* **or** *and* **and** *gates, and two types of fan-in zero gates:* **input** *gates, labeled with a variable x_i, and* **choice** *gates, labeled with a positive integer $1 \le i \le k$*[1].

Values propagate in an ND circuit as expected: the **input** and **choice** gates supply 0/1 values, a **not** gate outputs the negation of its input, an **and** gate outputs the conjunction of its inputs, and an **or** gate outputs the disjunction of its inputs. ND circuits with zero alternations coincide with the usual definition of a circuit; the only difference in the general case is the acceptance criterion, defined below:

Definition 4.2 *An ND circuit C with k alternations with the Σ_k* acceptance criterion *accepts an input x if and only if $\exists Y_1 \forall Y_2 \exists Y_3 \ldots QY_k$ for which $C(x) = 1$ when the CHOICE gates labeled with i take on the value Y_i (quantifier Q is $\exists$ if k is even and $\forall$ if k is odd). An ND circuit C with k alternations with the Π_k* acceptance criterion *accepts an input x if and only if the same circuit with the Σ_k acceptance criterion rejects x.*

For brevity, we use the phrase "Σ_k ND circuit[2]" in place of "non-deterministic circuit with k alternations and acceptance criterion Σ_k." A polynomially large Σ_k ND circuit is a non-uniform analog of Σ_k^{P} (and a polynomially large Π_k ND circuit is a non-uniform analog of Π_k^{P}), while a polynomially large $\Sigma_{\mathrm{poly}(n)}$ ND circuit is a non-uniform analog of PSPACE.

4.2 The basic problems

We first define the versions of the monotone minimum weight word problem for each level of the PH. We need one further definition:

Definition 4.3 *A* monotone set *is a subset $M \subseteq \{0,1\}^n$ with the property that $x \in M$ implies $x' \in M$, for all $x' \succeq x$.*

k-MONOTONE MINIMUM WEIGHT WORD (k-MMWW):

INSTANCE: A Π_{k-1} ND circuit C that accepts exactly a non-empty monotone set.

[1] Any complete basis will do; as will any constant fan-in.

[2] In another context, "Σ_k (and Π_k) circuits" refer to constant depth circuits with k alternations; the meaning is different here.

SOLUTION: A string x accepted by C.

MEASURE: The Hamming weight of x.

Observe that the decision version of k-MMWW is in Σ_k^{P}, because we can guess the string with few ones, and check whether C accepts it with a query to a Π_{k-1}^{P} oracle (which is powerful enough to evaluate a Π_{k-1} ND circuit).

A very similar scenario, when interpreted as a maximization problem, gives us the monotone maximum zeros problems; again, we define one version for each level of the PH:

k-MONOTONE MAXIMUM ZEROS (k-MMZ):

INSTANCE: A Π_{k-1} ND circuit C that accepts exactly a non-empty monotone set.

SOLUTION: A string x accepted by C.

MEASURE: The number of zeros in x.

Notice that it is not *a priori* clear that the problems, although complementary, have the same approximation hardness. In the next sections we show that they do, however this is not true, for example of similar complementary problems at the NP level.

4.3 Gap-producing reductions using or-codes

In this section we give the basic gap-producing reductions, showing that the monotone minimum weight word problems and the monotone maximum zeros problems are hard to approximation to within very large factors. The theorems that follow describe the important parameters of the reduction achieved given an or-code with particular parameters. We then derive a number of consequences by plugging in or-code constructions from Chapter 3.

We need the following lemma regarding the closure of Π_k^{P} under polynomially-bounded disjunction:

Lemma 4.1 *If $R(X)$ is a Π_k^p predicate, then $R'(X_1, X_2, \ldots X_n) = \bigvee_{i=1}^n R(X_i)$ is also a Π_k^p predicate. Furthermore, if k is even,*

$$\begin{aligned} R'(X_1, X_2, \ldots X_n) &\Leftrightarrow \bigvee_{i=1}^{n} (\forall Y_1)(\exists Y_2) \cdots (\exists Y_k) A(X_i, Y_1, Y_2, Y_3, \ldots Y_k) \\ &\Leftrightarrow (\forall Y_1^{(1)} Y_1^{(2)} \ldots Y_1^{(n)})(\exists Y_2)(\forall Y_3^{(1)} Y_3^{(2)} \ldots Y_3^{(n)}) \cdots (\exists Y_k) \\ &\quad \bigvee_{i=1}^{n} A(X_i, Y_1^{(i)}, Y_2, Y_3^{(i)}, \ldots Y_k), \end{aligned}$$

and if k is odd,

$$\begin{aligned} R'(X_1, X_2, \ldots X_n) &\Leftrightarrow \bigvee_{i=1}^{n} (\forall Y_1)(\exists Y_2) \cdots (\forall Y_k) A(X_i, Y_1, Y_2, Y_3, \ldots Y_k) \\ &\Leftrightarrow (\forall Y_1^{(1)} Y_1^{(2)} \ldots Y_1^{(n)})(\exists Y_2)(\forall Y_3^{(1)} Y_3^{(2)} \ldots Y_3^{(n)}) \cdots (\forall Y_k^{(1)} Y_k^{(2)} \ldots Y_k^{(n)}) \\ &\quad \bigvee_{i=1}^{n} A(X_i, Y_1^{(i)}, Y_2, Y_3^{(i)}, \ldots Y_k^{(i)}), \end{aligned}$$

where A is a predicate in P.

Proof: The first implication in each case follows from the quantifier characterization of Π_k^{P}. The second implication is straightforward manipulation of quantified expressions. ■

Theorem 4.2 *For every k, there is a reduction τ_k from* QSAT$_k$ *to k-*MMWW WITH GAP *g that requires an explicit or-code. On instance $\phi(X_1, X_2, \ldots X_k)$, using an explicit $(|X_1|, M, K, \epsilon)$ or-code with weight W, the gap is $g = (1-\epsilon)M/W$. Reduction τ_k runs in time* poly(s)*, and produces a Π_{k-1} ND circuit with M inputs and size* poly(s)*, where $s = |\phi| \cdot \max\{M, K\}$.*

Proof: Fix k, and let $\phi(x_1, x_2, \ldots x_k)$ be an instance of QSAT$_k$. Let **Enc** and **Dec** be the encoding and decoding functions for a $(|x_1|, M, K, \epsilon)$ or-code with weight W. We describe the instance of k-MMWW WITH GAP $(1-\epsilon)M/W$ by first describing a Π_{k-1}^{P} *algorithm* A. Then, using Lemma 1.4, we transform algorithm A into a Π_{k-1} ND circuit.

On input x, algorithm A has two main steps. First, it computes the set $\mathbf{Dec}(x)$. If $\mathbf{Dec}(x) = \{\text{FAIL}\}$, then A accepts. Otherwise, it proceeds to the second step. The second steps checks whether

$$(\exists x_1 \in \mathbf{Dec}(x))(\forall x_2)(\exists x_3) \cdots (Qx_k)\phi(x_1, x_2, x_3, \ldots x_k), \tag{4.1}$$

where Q is $\forall$ if k is even and $\exists$ if k is odd. The critical fact here is that $|\mathbf{Dec}(x)| \leq K$, so by Lemma 4.1, there is a Π_{k-1}^{P} algorithm that evaluates statement 4.1 above.

Claim 4.2.1 *Algorithm A accepts exactly a non-empty monotone set.*

Proof: Let x be an input accepted by A. If x has weight greater than $(1-\epsilon)M$, then clearly all $x' \succeq x$ are accepted by A. Otherwise, it must be that for some $y \in \mathbf{Dec}(x)$, $(\forall x_2)(\exists x_3)\cdots(Qx_k)\phi(y, x_2, x_3, \ldots x_k)$. By the properties of or-codes, for every $x' \succeq x$ of weight at most $(1-\epsilon)M$, we have $y \in \mathbf{Dec}(x')$, and hence A accepts. For $x' \succeq x$ of larger weight, A also accepts (because $\mathbf{Dec}(x') = \{\text{FAIL}\}$). This shows that A accepts exactly a monotone set, and because A accepts the all-ones string, A accepts exactly a non-empty monotone set. ∎

Claim 4.2.2 *If ϕ is a positive* QSAT$_k$ *instance, then A accepts a string x of weight W.*

Proof: Since ϕ is a positive instance, there exists some $y \in \{0,1\}^n$ for which

$$(\forall x_2)(\exists x_3)\cdots(Qx_k)\phi(y, x_2, x_3, \ldots x_k). \tag{4.2}$$

Take $x = \mathbf{Enc}(y)$. ∎

Claim 4.2.3 *If ϕ is a negative* QSAT$_k$ *instance, then the minimum weight string accepted by A has weight greater than $(1-\epsilon)M$.*

Proof: Suppose A accepted some string x of weight at most $(1-\epsilon)M$, then by the properties of or-codes, $\mathbf{Dec}(x) \neq \{\text{FAIL}\}$, and therefore there exists some $y \in \mathbf{Dec}(x)$ for which $(\forall x_2)(\exists x_3)\cdots(Qx_k)\phi(y, x_2, x_3, \ldots x_k)$. This contradicts ϕ being a negative QSAT$_k$ instance. ∎

Let $s = |\phi| \max\{M, K\}$. By the definition of explicit or-codes, $\mathbf{Dec}(x)$ runs in time $\text{poly}(\max\{M, K\})$; evaluating statement 4.1 in Π_{k-1}^{P} requires $\text{poly}(K|\phi|)$ time. Altogether, the running time of algorithm A is $\text{poly}(s)$.

By Lemma 1.4, we can build a Π_{k-1} ND circuit C of size $\text{poly}(s)$ from algorithm A. This circuit has M inputs, size $\text{poly}(s)$, and is constructible in time $\text{poly}(s)$. Circuit C accepts exactly the inputs accepted by algorithm A, and therefore, by the claims above, constitutes an instance of k-MMWW WITH GAP $(1-\epsilon)M/W$. ∎

We can use the same reduction to show that the gap-decision version of k-MMZ is Σ_2^{P}-hard to approximate to within very large factors. We obtain a similar theorem, whose gap relies on optimizing a different parameter of the or-code used in the reduction.

Theorem 4.3 *For every k, there is a reduction τ_k from* QSAT$_k$ *to k-*MMZ WITH GAP *g that requires an explicit or-code. On instance $\phi(X_1, X_2, \ldots X_k)$, using an explicit $(|X_1|, M, K, \epsilon)$ or-code with weight W, the gap is $g = \epsilon^{-1}(1 - \frac{W}{M})$. Reduction τ_k runs in time* $\mathrm{poly}(s)$, *and produces a Π_{k-1} ND circuit with M inputs and size* $\mathrm{poly}(s)$, *where $s = |\phi| \cdot \max\{M, K\}$.*

Proof: We use the same reduction as in the proof of Theorem 4.2 to produce Π_{k-1} ND circuit C. By Claim 4.2.2, if the QSAT$_k$ instance is a positive instance, then C accepts a word with weight W, which has $M - W$ zeros. By Claim 4.2.3, if the QSAT$_k$ instance is a negative instance, then C accepts no words of weight $(1 - \epsilon)M$ or less, which implies that the maximum number of zeros in a word accepted by C is $M - (1 - \epsilon)M = \epsilon M$. The gap is therefore $(M - W)/(\epsilon M)$, and the theorem follows. ■

Remark 4.1 *The reductions just given do not rely on the structure of* QSAT$_k$ *in any way. We could just as easily have reduced from other complete problems, or given a generic reduction from every problem in the class.*

Remark 4.2 *In a similar way, one can define a version of* MMWW *and* MMZ *for PSPACE, and show that a gap-decision variant of it is PSPACE-hard by reduction from* QBF, *using an identical proof technique. Because $\exists$PSPACE = PSPACE, this is not terribly interesting as an optimization problem (an "approximation" algorithm for a PSPACE optimization problem is powerful enough to solve the problem exactly); however there may be some application for the gap-decision versions of these problems in showing non-automizability of some simple propositional proof systems [10], a direction that we do not pursue here.*

Remark 4.3 *These reductions produce Π_{k-1} circuits with a stronger monotone property than is implied by the definitions. For example, let C be the* 3-MMWW *instance produced, and suppose C accepts X. Then for all settings Y of the universal choice gates, there exists a setting $Z_{Y,X}$ of the existential choice gates for which $C(X)$ evaluates to true. By monotonicity, we know that C also accepts all $X' \succeq X$. An examination of the proof of Theorem 4.2 above and Lemma 4.1 shows that for all settings Y of the universal choice gates, we may take the* same *setting of the existential choice gates, $Z_{Y,X}$, that depends only on Y and X, to obtain a setting for which $C(X')$ evaluates to true. A similar statement holds with regard to settings of the existential choice gates in the circuits produced in the reduction to k-*MMWW *for all $k \geq 3$.*

4.4 Self-improvement to amplify the gap

The technique of self-improvement is common in the hardness of approximation literature; see for example [35, Chapter 10]. We show in this section how to boost the gaps achieved in the previous section for k-MMWW by composing the circuits produced with themselves. This idea was used in a very similar setting by [2].

Lemma 4.4 *Given an instance of k-MMWW WITH GAP g on n variables, with size s, we can produce, in $O(n^\ell s)$ time, an instance of k-MMWW WITH GAP g^ℓ on n^ℓ variables, with size $O(n^\ell s)$.*

Proof: Let C be an instance of k-MMWW, so C is a Π_{k-1} ND circuit on n variables, with size s, that accepts exactly a non-empty monotone set. Define $C^{(\ell)}$ recursively as follows: $C^{(1)}$ is C, and $C^{(\ell)}$ is C with n independent copies of $C^{(\ell-1)}$ inserted at its inputs. It is clear that $C^{(\ell)}$ is a Π_{k-1} ND circuit; also $C^{(\ell)}$ accepts exactly a non-empty monotone set if C does.

For $C^{(2)}$, we need one copy of the original circuit C, plus n additional copies for each of the inputs. For $C^{(3)}$, we need n^2 additional copies; and so on. In general the size of $C^{(\ell)}$ is $(1 + n + n^2 + \ldots + n^{\ell-1})s$, which is at most $O(n^\ell s)$, as required. Also, it is clear that the number of variables of $C^{(\ell)}$ is n^ℓ.

To show that the gap is amplified as stated, we prove the following claim:

Claim 4.4.1 *C accepts a string of weight at most k if and only if $C^{(\ell)}$ accepts a string of weight at most k^ℓ.*

Proof: The forward direction is simple: if x is a string of weight at most k accepted by C, then we force the appropriate k copies of $C^{(\ell-1)}$ to output one so that the induced input to the topmost copy of C is x. By induction, each copy of $C^{(\ell-1)}$ can be made to output one with an input of weight at most $k^{\ell-1}$, so the total weight of the accepted string is at most k^ℓ.

In the other direction, suppose $C^{(\ell)}$ accepts a string x with weight at most k^ℓ. At the bottom level, $C^{(\ell)}$ has $n^{\ell-1}$ copies $C_1, C_2, \ldots C_{n^{\ell-1}}$ of the original circuit C. Consider the inputs $x_1, x_2, \ldots x_{n^{\ell-1}}$ to each of these copies, obtained by projecting x onto the appropriate subset of the input variables. If for some i, C_i rejects string x_i, then by monotonicity we can safely set all of the inputs to that C_i to false without changing the value of the output

#	problem	size measure N	reduction time	gap g
1	k-MMWW	# variables	poly$(sn^{\epsilon^{-1}})$	$N^{1/5-\epsilon}$
2	k-MMWW	circuit size	poly$(sn^{\epsilon^{-2}})$	$N^{1/5-\epsilon}$
3	k-MMWW	# variables	poly$(sn^{(\log n)\epsilon^{-1}})$	$N^{1-\epsilon}$
4	k-MMWW	circuit size	poly$(sn^{(\log n)\epsilon^{-2}})$	$N^{1-\epsilon}$
5	k-MMWW	# variables	poly$(sn^{\epsilon^{-1}})$ (*)	$N^{1-\epsilon}$
6	k-MMWW	circuit size	poly$(sn^{\epsilon^{-2}})$ (*)	$N^{1-\epsilon}$
7	k-MMZ	# variables	poly$(sn^{\epsilon^{-1}})$	$N^{1/8-\epsilon}$
8	k-MMZ	# variables	poly$(sn^{(\log n)\epsilon^{-1}})$	$N^{1-\epsilon}$
9	k-MMZ	# variables	poly$(sn^{\epsilon^{-1}})$ (*)	$N^{1-\epsilon}$

Table 4.1: Gap-producing reductions from QSAT$_k$ to k-MMWW and k-MMZ, in Theorem 4.5.

of $C^{(\ell)}$. In this way we obtain a string x' with weight at most k^ℓ, and its projections $x'_1, x'_2, \ldots x'_{n^{\ell-1}}$, with the property that each x'_i is either the all-zeros string or it is accepted by C_i. If some accepted x'_i has weight at most k, then we are done; otherwise, every x'_i that is not the all-zeros string has weight greater than k. Since the weight of x' is at most k^ℓ, there can be at most $k^{\ell-1}$ non-zero x'_i. Hence, x' induces an assignment to the inputs of $C^{(\ell-1)}$ with weight at most $k^{\ell-1}$, and we are done by induction. ■

Now, if C accepts a string of weight at most B then $C^{(\ell)}$ accepts a string of weight at most B^ℓ; if the minimum weight string C accepts has weight greater than gB, then the minimum weight string accepted by $C^{(\ell)}$ has weight greater than $(gB)^\ell$. Therefore the new gap is at least $(gB)^\ell/B^\ell = g^\ell$. ■

4.5 Putting it all together

The or-code constructions in Theorem 3.5, together with Theorem 4.2 and Lemma 4.4 yield reductions from QSAT$_k$ to k-MMWW with differing reduction complexities and approximation hardness ratios. Similarly, the or-code constructions together with Theorem 4.3 produce a range of hardness results for k-MMZ. We summarize these in the next theorem.

Theorem 4.5 *For every k, and $\epsilon > 0$, the reductions in Table 4.1 from* QSAT$_k$ *exist, where s is the size of the* QSAT$_k$ *instance, measured as the size of the DNF or CNF formula, and n is the number of variables quantified by the first existential quantifier. The entries marked with (*) use or-code number 3, 4, or 5 from Theorem 3.5. The reductions respectively* *succeed with high probability, run in expected time with an NP oracle, or are deterministic with a* Σ_2^p *oracle.*

Proof: For all of the reductions, we begin with an instance $\phi(X_1, X_2, \ldots X_k)$ of QSAT$_k$, with $|X_1| = n$, and $|\phi| = s$.

(Of 1.) Pick $M = n^c$ (the value of c will be specified below). By Theorem 3.5, we can construct in poly(n) time a $(n, n^c, O(n^{c/5-c_1}), 1/2)$ or-code with weight $W = O(n^{4/5c+c_1+1})$, where c_1 is the constant in the statement of the theorem. By Theorem 4.2, we can use this or-code in a reduction that produces an instance of k-MMWW WITH GAP g, where $g = \frac{1}{2}M/W$. Let N be the number of variables, so $N = M = n^c$. Picking $c > \epsilon^{-1}(c_1 + 1)$, we have:

$$g = \Omega(M/W) = \Omega(n^{c/5+c_1+1}) > N^{1/5-\epsilon},$$

for sufficiently large n. Altogether the reduction runs in poly$(sN) =$ poly$(sn^{\epsilon^{-1}})$ times.

(Of 2.) The circuit size of the instance produced in (1) is $N' = N^\gamma$ for some constant γ. Applying Lemma 4.4, with $\ell = \epsilon^{-1}\gamma/5$, we obtain an instance of k-MMWW WITH GAP g' of size $N^{\ell+\gamma}$, with $g' = g^\ell$, and

$$g' > N^{(1/5-\epsilon)\ell} > (N')^{1/5-2\epsilon}.$$

Therefore, taking $\epsilon' = 2\epsilon$, we can obtain the desired gap in terms of N', the size of the circuit, in poly$(sN) \cdot N^\ell =$ poly$(sn^{\epsilon^{-2}})$ time.

(Of 3.) Pick $M = 2^{c\log^2 n}$, for some c that will be specified below, and let $\alpha > 1$ be any constant. Using Theorem 3.5, construct in poly(n) time, a $(n, M, M^\alpha, 1/2)$ or-code with weight $W = 2^{\gamma(\log^2 n+1)}$ for a constant γ. By Theorem 4.2, we can use this or-code in a reduction that produces an instance of k-MMWW WITH GAP g, where $g = \frac{1}{2}M/W$. The number of variables is $N = M = 2^{c\log^2 n}$. Picking $c > \epsilon^{-1}\gamma$, we have:

$$g = \Omega(M/W) = \Omega(2^{c\log^2 n-\gamma(\log^2 n+1)}) > N^{1-\epsilon},$$

for sufficiently large n. Altogether the reduction runs in poly$(sN) =$ poly$(sn^{(\log n)\epsilon^{-1}})$ time.

(Of 4.) Using Lemma 4.4 in the same way as (2) above, we obtain an instance of k-MMWW WITH GAP g' with size N' and $g' \geq (N')^{1-2\epsilon}$. The reduction runs in time poly$(sn^{(\log n)\epsilon^{-2}})$.

(Of 5.) Pick $M = n^c$ for some c that will be specified below. By Theorem 3.5, construct in poly(M, n) time (as noted in the Theorem, the running time may be probabilistic, or require an NP or Σ_2^{P} oracle), a $(n, M, M, 1/2)$ or-code with weight $W = O(n^2)$. By Theorem 4.2, we can use this or-code in a reduction that produces an instance of k-MMWW WITH GAP g, where $g = \frac{1}{2}M/W$. The number of variables is $N = M = n^c$. Picking $c > 2\epsilon^{-1}$, we have:

$$g = \Omega(M/W) = \Omega(n^{c-2}) > N^{1-\epsilon},$$

for sufficiently large n. Altogether the reduction runs in $\mathrm{poly}(sN) = \mathrm{poly}(sn\epsilon^{-1})$ time.

(Of 6.) Using Lemma 4.4 in the same way as (2) above, we obtain an instance of k-MMWW WITH GAP g' with size N' and $g' \geq (N')^{1-2\epsilon}$. The reduction runs in time $\mathrm{poly}(sn^{\epsilon^{-2}})$.

(Of 7.) Let c_1 be the constant from Theorem 3.5 (1). Pick $M = n^{c'}$ and $\delta = n^{-c}$, for c and c' to be determined later. By Theorem 3.5, construct in $\mathrm{poly}(n)$ time, a $(n, M, O(n^{c'/5+2c/5-c_1}), \delta)$ or-code with weight $W = O(n^{4c'/5+8c/5+c_1+1})$. By Theorem 4.3, we can use this or-code in a reduction that produces an instance of k-MMZ WITH GAP g, where $g = \delta^{-1}(1 - W/M)$. The number of variables is $N = M$. Now, pick c' in terms of c so that $W/M = 1/n$, i.e., pick $c' = 8c + c_1/5 + 2/5$. Finally, pick $c > \gamma\epsilon{-}1$, for some γ that depends on c_1, and we have:

$$g = \delta^{-1}(1 - O(1/n)) = (1 - o(1))n^c > N^{1/8-\epsilon},$$

for sufficiently large n. Altogether the reduction runs in $\mathrm{poly}(sN) = \mathrm{poly}(sn^{\epsilon^{-1}})$ time.

(Of 8.) Let c_1 be the constant from Theorem 3.5 (1). Pick $M = 2^{(c+1)\log^2 n}$ and $\delta = 2^{-c\log^2 n}$, for c to be determined later, and let $\alpha = 2^{c_1}$. By Theorem 3.5, construct in $\mathrm{poly}(n)$ time, a (n, M, M^α, δ) or-code with weight $W = O(n2^{c\log^2 n})$. By Theorem 4.3, we can use this or-code in a reduction that produces an instance of k-MMZ WITH GAP g, where $g = \delta^{-1}(1 - W/M)$. The number of variables is $N = M$. Now, pick $c > \epsilon{-}1$, and we have:

$$g = \delta^{-1}(1 - W/M) = (1 - o(1))2^{c\log^2 n} > N^{1-\epsilon},$$

for sufficiently large n. Altogether the reduction runs in $\mathrm{poly}(sN) = \mathrm{poly}(sn^{\epsilon^{-1}})$ time.

(Of 9.) Pick $M = n^{c+3}$ and $\delta = n^{-c}$. By Theorem 3.5, construct in $\mathrm{poly}(M, n^c)$ time (as noted in the Theorem, the running time may be probabilistic, or require an NP or Σ_2^{P} oracle), a (n, M, M, δ) or-code with weight $W = O(n^{2+c})$. By Theorem 4.3, we can use this or-code in a reduction that produces an instance of k-MMZ WITH GAP g, where

$g = \delta^{-1}(1 - W/M)$. The number of variables is $N = M = n^{c+3}$. Picking $c > 3\epsilon^{-1}$, we have:

$$g = \delta^{-1}(1 - O(1/n)) = (1 - o(1))n^c > N^{1-\epsilon},$$

for sufficiently large n. Altogether the reduction runs in $\text{poly}(sN) = \text{poly}(sn^{\epsilon^{-1}})$ time. ■

Because the self-improvement technique seems to work only for k-MMWW and not k-MMZ, we cannot achieve gaps for k-MMZ in terms of the circuit size that match the gaps expressed in terms of the number of variables. However, the size of the circuit produced in any of the reductions described in the previous theorem is always polynomial in the number of variables, so in particular we can achieve a gap (in terms of the circuit size N) of N^ϵ for some $\epsilon > 0$ in $\text{poly}(sn)$ time.

Chapter 5

Inapproximable Problems in the Second Level of the PH

In this chapter, we define a variety of natural optimization problems in the PH, and give approximation-preserving reductions from the appropriate version of k-MMWW or k-MMZ to these problems. Many of these are problems relating to circuit minimization (including MIN DNF and some variants, the SHORTEST IMPLICANT CORE problem from Chapter 2, and a substep in logic minimization heuristics called IRREDUNDANT); some are more basic combinatorial optimization problems that are variants of more familiar NP optimization problems (including SUCCINCT SET COVER and two succinct graph problems). A summary of some of the inapproximability results we obtain for these problems may be found in Table 5.1 at the end of this chapter.

5.1 Shortest implicant core

This problem was introduced and shown to be Σ_2^p-complete in Chapter 2. Here we use the same ideas to give a gap-preserving reduction from 2-MMWW. We define here the optimization version of the problem:

SHORTEST IMPLICANT CORE:

INSTANCE: A DNF formula ϕ and an implicant C of ϕ.

SOLUTION: An implicant $C' \subseteq C$ of ϕ

MEASURE: The length of C': $|C'|$.

In the context of circuit minimization, SHORTEST IMPLICANT CORE is a formalization of a sub-step (called "expand") used widely in heuristics for DNF minimization [23]. The purpose of this step is to remove extraneous literals from the terms of the DNF (hence turning every term into a prime implicant and *expanding* the subcube covered by that term). Underlying this procedure is an optimization problem, repeated once for each term in the DNF, that is exactly SHORTEST IMPLICANT CORE; to expand a DNF ϕ, take C to be each term t of ϕ in succession, compute the shortest implicant core C' given instance $\langle \phi, C \rangle$, and replace t withC'. We proceed with the reduction:

Theorem 5.1 *There is a polynomial-time gap-preserving reduction from* 2-MMWW WITH GAP g *to* SHORTEST IMPLICANT CORE WITH GAP g *that maps instances of* 2-MMWW *with* n *variables and circuit size* m *to instances* SHORTEST IMPLICANT CORE *with* $O(m)$ *variables and* $O(m)$ *terms.*

Proof: Let Π_1 ND circuit D be an instance 2-MMWW, with n input variables $X = (x_1, x_2, \ldots x_n)$, and size m. We first replace the choice gates with variables Y. The resulting circuit D' is an ordinary deterministic circuit of size m with the property that

$$D \text{ accepts } X \Leftrightarrow (\forall Y) D'(X, Y). \tag{5.1}$$

We now apply Lemma 1.3 to circuit D' to obtain a 3-DNF $\phi(X, Y, Z)$ of size $O(m)$. By Equation 5.1 and the lemma, we have:

$$D \text{ accepts } X \Leftrightarrow (\forall Y) D'(X, Y) \Leftrightarrow (\forall Z)(\forall Y) \phi'(X, Y, Z). \tag{5.2}$$

We take as our instance of SHORTEST IMPLICANT CORE the 3-DNF ϕ and the implicant $C = x_1 x_2 \ldots x_n$. We know C is an implicant of ϕ' because D accepts the all-ones string.

Claim 5.1.1 Π_1 *ND circuit* D *accepts a string* X' *with weight* k *if and only if there exists an implicant* $C' \subseteq C$ *of* ϕ *with* $|C'| = k$.

Proof: ($\Rightarrow$) Let $C' \subseteq x_1 x_2 \ldots x_n$ consist of exactly those x_i that are set to one in string X'. By equation 5.2, $(\forall Y, Z) \phi(X', Y, Z) = 1$. Moreover, every string $X'' \succeq X'$ is accepted by D by monotonicity; therefore $(\forall Y, Z) \phi(X'', Y, Z) = 1$ for all such strings. Therefore C' is an implicant of ϕ', of size k.

($\Leftarrow$) Given C', let X' be the string with a 1 in position i if and only if $x_i \in C'$. Since C' is an implicant and X' is consistent with C', it must be that $(\forall Y, Z)\phi(X', Y, Z)$. By Equation 5.2 D accepts X', and the weight of X' is k. ■

As required, ϕ has $O(m)$ terms and $O(m)$ variables, and by the above claim the gap g is exactly preserved. ■

Remark 5.1 *The proof of Theorem 5.1 shows that the problem remains hard when the DNF in* SHORTEST IMPLICANT CORE *is a 3-DNF.*

5.2 Maximum literal deletion

The maximization version of SHORTEST IMPLICANT CORE is the problem defined below:

MAXIMUM LITERAL DELETION:

INSTANCE: A DNF formula ϕ and an implicant C of ϕ.

SOLUTION: A subset $D \subseteq C$ of the literals in C to delete, such that $C - D$ is an implicant of ϕ.

MEASURE: The size of D: $|D|$.

Theorem 5.2 *There is a polynomial-time gap-preserving reduction from* 2-MMZ WITH GAP *g to* MAXIMUM LITERAL DELETION WITH GAP *g that maps instances of* 2-MMZ *with n variables and circuit size m to instances* MAXIMUM LITERAL DELETION *with $O(m)$ variables and $O(m)$ terms.*

Proof: We use the same transformation from a Π_2 ND circuit D to a DNF ϕ and an implicant C of ϕ as in the proof of Theorem 5.1 above. A simple rephrasing the statement of Claim 5.1.1 shows that D accepts a string with k zeros if and only if there exist k literals that can be deleted from C, and therefore the gap g is exactly preserved. ■

5.3 Succinct set cover

We now turn to a more basic combinatorial problem, the succinct version of the ubiquitous NP optimization problem SET COVER.

SUCCINCT SET COVER:

INSTANCE: A collection $\mathcal{C} = \{C_1, C_2, \ldots C_m\}$ of Boolean circuits over n variables.

SOLUTION: A subset $\mathcal{C}' \subseteq \mathcal{C}$ for which $(\vee_{C_i \in \mathcal{C}'} C_i) \equiv 1$.

MEASURE: The cardinality of the subset: $|\mathcal{C}'|$

Many of the problem relating to DNF minimization, including MIN DNF itself can be seen as a sort of succinct set cover problem in the following sense: the classic exact algorithm for DNF minimization (the Quine-McCluskey algorithm [67, 56]) generates the set of all prime implicants of the input DNF ϕ, and then solves the resulting covering problem; that is, it selects the smallest collection of prime implicants that cover all the ones of ϕ. The generated covering problem may in general be exponentially large; we can therefore think of the input DNF as representing this problem *succinctly*. The SUCCINCT SET COVER problem defined above differs from this scenario by having only polynomially many sets, but the ground set is exponentially large, as in the prime implicant covering problem.

Theorem 5.3 *There is a polynomial-time gap-preserving reduction from* 2-MMWW WITH GAP g *to* SUCCINCT SET COVER WITH GAP $\Omega(g)$ *that maps instances of* 2-MMWW *with* n *variables,* c *choice gates, and circuit size* s *to instances* SUCCINCT SET COVER *with* $n + c$ *variables,* $n + 1$ *sets, and total circuit size* $O(s)$.

Proof: Let Π_1 ND circuit D be an instance 2-MMWW, with n input variables $X = (x_1, x_2, \ldots x_n)$, c choice gates, and size s. As in the proof of Theorem 5.1, we first replace the choice gates with variables Y to obtain circuit D' of size s on $n + c$ variables such that

$$D \text{ accepts } X \Leftrightarrow (\forall Y) D'(X, Y). \tag{5.3}$$

The instance of SUCCINCT SET COVER is given by $\mathcal{C} = \{D', \overline{x_1}, \overline{x_2} \ldots \overline{x_n}\}$.

Claim 5.3.1 Π_1 *ND circuit* D *accepts a string* X' *with weight* k *if and only if there exists a subset* $\mathcal{C}' \subseteq \mathcal{C}$ *of size* $k+1$ *for which* $\bigvee_{C_i \in \mathcal{C}'} C_i \equiv 1$.

Proof: ($\Rightarrow$) Let $\mathcal{C}' \subseteq \mathcal{C}$ be D' together with the k circuits $\overline{x_i}$ for each x_i set to true in X'. We show that $\bigvee_{C_i \in \mathcal{C}'} C_i \equiv 1$. For an arbitrary X, Y, we see that either (1) $X \succeq X'$, in which case $D'(X,Y) = 1$ by monotonicity, or (2) there exists an x_i set to zero by X and one by X', in which case $\overline{x_i}(X,Y) = 1$ is true. Since X and Y were arbitrary, $\mathcal{C}'$ is indeed a cover, of size $k+1$.

($\Leftarrow$) We note that $\mathcal{C}'$ must include D', which is the only circuit in $\mathcal{C}$ that covers the all-ones assignment. Let X' be the assignment that sets x_i to one if and only if the circuit $\overline{x_i}$ is in $\mathcal{C}'$. We argue that D accepts X'. If not, then there exists a Y for which $D'(X',Y) = 0$; but then every circuit in $\mathcal{C}'$ evaluates to false on X', Y, contradicting the assumption that $\bigvee_{C_i \in \mathcal{C}'} C_i \equiv 1$. ■

As promised, there are $n+1$ circuits in $\mathcal{C}$, each over $n+c$ variables, with total size $O(s)$, and by the above claim, the gap g is preserved within a constant factor. ■

It is a simple matter to transform the circuit D' in the instance of SUCCINCT SET COVER generated above into a 3-DNF formula using Lemma 1.3. As all of the other circuits are single literals, we can easily obtain an almost identical reduction to the 3-DNF version of SUCCINCT SET COVER, defined below:

3-DNF SUCCINCT SET COVER:

INSTANCE: A collection $\mathcal{C} = \{\phi_1, \phi_2, \ldots \phi_m\}$ of 3-DNF formulae over n variables.

SOLUTION: A subset $\mathcal{C}' \subseteq \mathcal{C}$ for which $(\vee_{\phi_i \in \mathcal{C}'} \phi_i) \equiv 1$.

MEASURE: The cardinality of the subset: $|\mathcal{C}'|$

Theorem 5.4 *There is a polynomial-time gap-preserving reduction from* 2-MMWW WITH GAP g *to* 3-DNF SUCCINCT SET COVER WITH GAP $\Omega(g)$ *that maps instances of* 2-MMWW *with* n *variables, and circuit size* s *to instances* 3-DNF SUCCINCT SET COVER *with* $O(s)$ *variables,* $n+1$ *sets, and total size of the DNFs* $O(s)$.

Proof: The proof is a simple modification, sketched above, of the proof of Theorem 5.3. ■

Remark 5.2 *For both* SUCCINCT SET COVER *and* 3-DNF SUCCINCT SET COVER, *if we take* $\epsilon \leq 1/2$ *in the or-code used in the reduction from* QSAT_2 *to* 2-MMWW *the (succinct) sets each cover at least half of the elements of the ground set. This is clear for the sets that are simply negated literals; for the set which encodes the* Π_1 *ND circuit* C*, we note that* C *accepts more than half of its inputs if* $\epsilon \leq 1/2$ *(it accepts all inputs* x *with a majority of ones, and that this implies that the 3-DNF derived from* C *covers at least half of the elements in the ground set.*

5.4 Minimum term equivalent DNF formula

We now prove that DNF minimization is Σ_2^p-complete, and hard to approximate within very large factors. There are a few variants of the DNF minimization problem; one was already defined in Chapter 2. Before we return to that problem, we consider the following variant, which measures the size of the DNF by the number of terms in the formula:

MINIMUM TERM EQUIVALENT DNF FORMULA (MIN TERM DNF):

INSTANCE: A DNF formula ϕ.

SOLUTION: A DNF formula ϕ' for which $\phi' \equiv \phi$.

MEASURE: The number of terms in ϕ'.

Theorem 5.5 *There is a polynomial-time gap-preserving reduction from* SHORTEST IMPLICANT CORE WITH GAP g *to* MIN TERM DNF WITH GAP $\Omega(g)$ *that maps instances of* SHORTEST IMPLICANT CORE *with implicant* C *of size* n*,* m *variables, and* s *terms to instances of* MIN TERM DNF *with* $O(m+s)$ *variables,* $O(n^2 s)$ *terms, and* $O((ns)^2 \log(ns))$ *occurrences of literals.*

Proof: Let DNF ϕ and implicant C of ϕ be an instance of SHORTEST IMPLICANT CORE, with $|C| = n$, and ϕ having m variables and s terms. The reduction builds on the reduction to 3-DNF SUCCINCT SET COVER, and uses two additional ideas: first, we force any equivalent DNF to include certain terms (intuitively, the ones that constitute the single multi-term DNF in the 3-DNF SUCCINCT SET COVER instance) by adding variables. The only choice to be made in forming an equivalent DNF is then which of the remaining terms to include (intuitively, the remaining terms will be the single-literal DNFs in the 3-DNF SUCCINCT SET COVER instance). Second, we weight these additional terms (by taking the conjunction of each with a formula that requires many terms to express in DNF) so that the number of these terms included in the equivalent DNF dominates its size, instead of the "forced" terms.

We now describe the DNF that the reduction produces more precisely. Let $t_1, t_2, \ldots t_s$ be the terms of ϕ, and let $x_1, \ldots x_n$ be the variables in C, which we assume to be positive, without loss of generality. Define t_{ij} as follows:

$$t_{ij} = (t_i)\overline{x_j} z_1 z_2 \ldots z_{i-1}\ z_{i+1} \ldots z_s \tag{5.4}$$

Some of the t_{ij} terms may the null term – if they contain both x_i and $\overline{x_i}$ for some i. Let $s' \leq ns$ be the number of non-null terms among the t_{ij}, and define $r = \lceil \log 2s' \rceil$. Let p_i be the i-th term of the (unique) 2^{r-1}-term DNF representation of $\bigoplus(a_1, a_2, \ldots a_r)$, where the a_i are new variables. Our instance of MIN TERM DNF is:

$$\begin{aligned} \phi' &= \bigvee_{i=1}^{s} \bigvee_{j=1}^{n} t_{ij} \vee \bigvee_{i=1}^{s'} \bigvee_{j=1}^{n} u_{ij}, & (5.5) \\ \text{where } u_{ij} &= (p_i)\overline{x_j} z_1 z_2 \ldots z_s & (5.6) \end{aligned}$$

and the z_i are new variables. Notice that ϕ' has $O(m+s)$ variables, $O(n^2 s)$ terms, and $O((ns)^2 \log(ns))$ occurrences of literals.

For ease of exposition, let us give a name to the "first half" of ϕ': define T to be $\bigvee_{i=1}^{s} \bigvee_{j=1}^{n} t_{ij}$. It will also be useful to define the function P (which is different from $\bigoplus(a_1, a_2, \ldots a_r)$ if and only if $s' < 2^{r-1}$) as follows: $P(a_1, a_2, \ldots a_r) = \bigvee_{i=1}^{s'} p_i$. We can now prove the following useful fact:

Claim 5.5.1 *Let A be an assignment that sets at least one x variable to false and all z variables to true. Then $\phi(A) \Leftrightarrow T(A)$.*

Proof: Let R be the restriction that sets all z variables to true, and notice that $T_R \equiv \phi \wedge \bigvee_{j=1}^{n} \overline{x_j}$. The claim follows. ∎

Claim 5.5.2 *If there exists an implicant $C' \subseteq C$ of ϕ of size at most k then ϕ' has an equivalent DNF ϕ'' with at most $s'(k+1)$ terms.*

Proof: Given C' as specified, we construct DNF ϕ'' below and show that it is equivalent to ϕ'.

$$\phi'' = \bigvee_{i=1}^{s} \bigvee_{j=1}^{n} t_{ij} \vee \bigvee_{i=1}^{s'} \bigvee_{x_j \in C'} u_{ij} \tag{5.7}$$

Since ϕ'' consists of a subset of the terms of ϕ' we know that $\phi'' \rightarrow \phi'$; to prove equivalence, we only need to verify that $\phi' \rightarrow \phi''$. Let A be an assignment for which $\phi'(A) = 1$. Since ϕ' covers no assignments with all x variables true, A must set at least one x variable to false. If A sets some z variable to zero, or $P(A) = 0$, then A must be covered by T, since the u_{ij} terms do not cover any such assignments. As ϕ'' is the disjunction of T and additional terms, $\phi''(A) = 1$. Otherwise, A sets all of the z variables to true, and $P(A) = 1$. In this case, either A is consistent with C', which implies $\phi(A) = 1$ and $T(A) = 1$ by Claim 5.5.1, or else A sets some $x_j \in C'$ to false and then one of u_{ij} terms included in ϕ'' covers it. In all cases, $\phi''(A) = 1$ if $\phi'(A) = 1$; hence $\phi'' \equiv \phi'$. Formula ϕ'' has $s' + s'k = s'(k+1)$ terms as required. ■

Claim 5.5.3 *If ϕ' has an equivalent DNF ϕ'' with at most $s'(k+1)$ terms, then there exists an implicant $C' \subseteq C$ of ϕ of size at most $\lceil k \rceil$.*

Proof: Suppose ϕ'' is a DNF equivalent to ϕ' with at most $s'(k+1)$ terms. Our goal is to construct, using ϕ'', an implicant $C' \subseteq C$ of ϕ of size at most $\lceil k \rceil$.

As a first step, we partition the terms of ϕ'' into two sets S_1 and S_2: S_1 contains every term of ϕ'' that contains some p_i (for $1 \leq i \leq s'$) in its entirety; S_2 contains all other terms. We make two observations about the terms in S_2:

Fact 1: Every term $t \in S_2$ implies T. Suppose not. Then there is an assignment A consistent with t for which $T(A) = 0$. Because t does not contain any p_i in its entirety, we can construct assignment A' by modifying the a variables in A so that $P(A') = 0$. Because T does not depend on the a variables, $T(A') = 0$. But $T(A') = 0$ and $P(A') = 0$ implies $\phi''(A') = 0$, which contradicts t being a term of ϕ''.

Fact 2: There are at least s' distinct terms in S_2. For each non-null t_{ij}, let A_{ij} be an assignment consistent with t_{ij} and with z_i mapped to false and the a variables set so that $P(A_{ij}) = 0$. Some term t'_{ij} of ϕ'' covers A_{ij}, and it is clear that t'_{ij} must include the literal z_ℓ for all $\ell \neq i$, for otherwise t'_{ij} would cover an assignment with two z

variables mapped to false, and ϕ'' covers no such assignments. Additionally, t'_{ij} does not contain the literal z_i, because it covers assignment A_{ij} which has z_i mapped to false. We conclude that the terms t'_{ij} must be pairwise distinct, and all of them are in S_2, because $P(A_{ij}) = 0$.

We now restrict our attention to the terms in S_1. Each of these terms contains in its entirety some p_i for $1 \leq i \leq s'$; also, each remaining term contains some negated x variables (because ϕ'' does not cover any assignment with all x variables true). Therefore, we can label each term $t \in S_1$ with a pair (i, j) so that t contains p_i the literal $\overline{x_j}$. By a pigeonhole argument, there must be some i_0 for which no more than $\lceil |S_1|/s' \rceil$ $(i_0, \cdot)$-labeled terms are present. We include in C' all literals x_j for which an (i_0, j)-labeled term is present.

We now show that C' is an implicant of ϕ. If it is not, then there exists some assignment A consistent with C' for which ϕ is false. Since C is an implicant of ϕ, this A must be inconsistent with C, and hence A sets some x variable to false. Extend assignment A so that it sets all z variables to true and the a variables so that they are consistent with p_{i_0}, obtaining assignment A'. In the next paragraph, we argue that $\phi'(A) = 1$, but $\phi''(A) = 0$, contradicting $\phi' \equiv \phi''$.

By construction, $\phi'(A) = 1$, because ϕ' covers all assignments consistent with some p_i, with some x variable set to false, and with all z variables set to true. However, by our choice of A, $\phi(A) = 0$, which implies $T(A) = 0$ by Claim 5.5.1, which implies that A falsifies every term in S_2, by Fact 1. Furthermore, A falsifies every term t in S_1, because it either sets some x variable to false that appears in term t, or else t contains p_i, $i \neq i_0$ in its entirety, and A is consistent with p_{i_0}. Since ϕ'' comprises exactly the terms in S_1 and S_2, $\phi''(A) = 0$, and the contradiction is reached. We conclude that C' must be an implicant of ϕ.

Finally, as required:

$$|C'| \leq \left\lceil \frac{|S_1|}{s'} \right\rceil \leq \left\lceil \frac{s'(k+1) - s'}{s'} \right\rceil = \lceil k \rceil. \tag{5.8}$$

■

Claim 5.5.2 and 5.5.3 show that the reduction preserves the gap to within a constant factor; this completes the reduction from SHORTEST IMPLICANT CORE to MIN TERM DNF. ■

5.5 Minimum equivalent DNF formula

We now return to the DNF minimization problem we saw in Chapter 2, in which the size of the DNF is measured by the number of occurrences of literals. The optimization problem is defined below:

MINIMUM EQUIVALENT DNF FORMULA (MIN DNF):

INSTANCE: A DNF formula ϕ.

SOLUTION: A DNF formula ϕ' for which $\phi' \equiv \phi$.

MEASURE: The number of occurrences of literals in ϕ'.

Using essentially the same reduction as we used for MIN TERM DNF, we show that MIN DNFis Σ_2^p-complete and hard to approximate to within very large factors.

Theorem 5.6 *There is a polynomial-time gap-preserving reduction from* SHORTEST IMPLICANT CORE WITH GAP g *to* MIN DNF WITH GAP $\Omega(g)$ *that maps instances of* SHORTEST IMPLICANT CORE *with implicant C of size n, m variables, and s terms to instances of* MIN TERM DNF *with $O(m+s)$ variables, $O(n^2 s)$ terms, and $O((ns)^2 \log(ns))$ occurrences of literals.*

Proof: We use the same reduction as in the proof of Theorem 5.5. Given an instance of SHORTEST IMPLICANT CORE that is given by 3-DNF ϕ (recall Remark 5.1) and implicant C we produce an instance ϕ' of MIN DNF, as given in Equation 5.6. We prove the following modified versions of Claims 5.5.2 and 5.5.3.

Claim 5.6.1 *If there exists an implicant $C' \subseteq C$ of ϕ of size at most k then ϕ' has an equivalent DNF ϕ'' with at most $s'[(s+3)+k(s+r+1)]$ occurrences of literals.*

Proof: We simply count the number of literals in formula ϕ'' given in Equation 5.7 of the proof of Claim 5.5.2. ■

Claim 5.6.2 *If ϕ' has an equivalent DNF ϕ'' with at most $s'[(s+2)+k(s+r)]$ occurrences of literals, then there exists an implicant $C' \subseteq C$ of ϕ of size at most $\lceil k \rceil$.*

Proof: We follow the proof of Claim 5.5.3. We first divide the terms of ϕ'' into sets S_1 and S_2. We can add to the reasoning under Fact 2 that each term t'_{ij} must also include each of the 3 literals in t_i. If it excluded any one of these literals, then it would cover an assignment inconsistent with $t_i \wedge \bigvee_{j=1}^{n} \overline{x_j}$ with z_i set to false, and that falsifies P; ϕ' covers no such assignments. Therefore S_2 contains at least s' terms, each of which contains at least $s+2$ literals.

Now, turning to the terms in S_1, we already know that every term in S_1 contains some p_i in its entirety, and some negated x variable, accounting for $r+1$ literals. In addition, every term in S_1 (indeed in all of ϕ'') must contain at least $s-1$ positive z literals; otherwise, it covers some assignment with two z variables set to false, and ϕ' covers no such assignments. Therefore, every term in S_1 must have at least $s+r$ literals.

Therefore, the terms in S_2 account for at least $s'(s+2)$ occurrences of literals, leaving at most $s'k(s+r)$ occurrences of literals for the terms in S_1. Since each term of S_1 must have at least $s+r$ literals, we conclude that $|S_1| \leq s'k$. Now, continuing with the proof of Claim 5.5.3, the constructed implicant C' has length bounded by:

$$|C'| \leq \left\lceil \frac{|S_1|}{s'} \right\rceil \leq \left\lceil \frac{s'k}{s'} \right\rceil = \lceil k \rceil. \tag{5.9}$$

■

The two claims above show that the reduction preserves the gap g to within a constant factor; this concludes the reduction from SHORTEST IMPLICANT CORE to MIN DNF.

■

5.6 Irredundant

In addition to SHORTEST IMPLICANT CORE ("expand"), there is a procedure referred to as "irredundant" that is also a sub-step used widely in heuristics for DNF minimization [23]. The purpose of this step is to remove extraneous terms from a DNF, in contrast to "expand," which removes extraneous literals from a term. The goal of irredundant is to obtain the DNF with the fewest terms, and it is not hard to see that the greedy approach fails here. In fact, we show in this section that the optimization problem IRREDUNDANT is just as hard as DNF minimization, in terms of its complexity and approximation hardness. First, the problem definition:

IRREDUNDANT:

INSTANCE: A DNF formula $\phi = t_1 \vee t_2 \vee \cdots \vee t_n$.

SOLUTION: A subset $I \subseteq [1 \ldots n]$ for which $\phi \equiv \bigvee_{i \in I} t_i$.

MEASURE: The cardinality of I: $|I|$.

Theorem 5.7 *There is a polynomial-time gap-preserving reduction from* SHORTEST IMPLICANT CORE WITH GAP g *to* IRREDUNDANT WITH GAP $\Omega(g)$ *that maps instances of* SHORTEST IMPLICANT CORE *with implicant C of size n, m variables, and s terms to instances of* IRREDUNDANT *with $O(m+s)$ variables, $O(n^2 s)$ terms, and $O((ns)^2 \log(ns))$ occurrences of literals.*

Proof: We use the same reduction as in the proof of Theorem 5.5. Given an instance of SHORTEST IMPLICANT CORE that is given by DNF ϕ and implicant C we produce an instance ϕ' of IRREDUNDANT, as given in Equation 5.6.

We observe that the equivalent DNF ϕ'' given in Equation 5.7 consists only of terms from DNF ϕ' given in Equation 5.6. We can conclude that essentially Claim 5.5.2 holds: if there exists an implicant $C' \subseteq C$ of ϕ of size at most k then there exists a subset of at most $s'(k+1)$ terms of ϕ' that are equivalent to ϕ'.

In the other direction, we note that the existence of an equivalent DNF containing at most $s'(k+1)$ terms of ϕ' trivially implies the existence of a DNF ϕ'' equivalent to ϕ' and containing at most $s'(k+1)$ terms; we now apply Claim 5.5.3 to see that this implies the existence of an implicant C of ϕ with size at most $\lceil k \rceil$. ∎

5.7 Maximum term deletion

The maximization version of IRREDUNDANT asks for the largest number of terms that may be deleted from a DNF without altering the function computed.

MAXIMUM TERM DELETION:

INSTANCE: A DNF formula $\phi = t_1 \vee t_2 \vee \cdots \vee t_n$.

SOLUTION: A subset $D \subseteq [1 \ldots n]$ for which $\phi \equiv \bigvee_{i \notin D} t_i$.

MEASURE: The cardinality of D: $|D|$.

Theorem 5.8 *There is a polynomial-time gap-preserving reduction from* MAXIMUM LITERAL DELETION WITH GAP g *to* MAXIMUM TERM DELETION WITH GAP $\Omega(g)$ *that maps instances of* MAXIMUM LITERAL DELETION *with implicant C of size n, m variables, and s terms to instances of* MAXIMUM TERM DELETION *with $O(m+s)$ variables, $O(n^2 s)$ terms, and $O((ns)^2 \log(ns))$ occurrences of literals.*

Proof: We use the same transformation as in the proof of Theorem 5.5. Given an instance of MAXIMUM LITERAL DELETION that is given by DNF ϕ and implicant C we produce an instance ϕ' of MAXIMUM TERM DELETION, as given in Equation 5.6.

An examination of Claim 5.5.2 reveals that it also proves, in the present setting, that if a set D of at least k literals may be deleted from C, then there exists a DNF ϕ'' equivalent to ϕ' that may be obtained by deleting at least $s'k$ terms from ϕ'.

Similarly, a restatement of Claim 5.5.3 shows that if a set of $s'k$ terms can be deleted from ϕ' without changing the function computed, then $\lfloor k \rfloor$ literals can be deleted from implicant C.

Together, these modified claims show that the gap is preserved to within constant factors. ■

5.8 Succinct rich hypergraph cover

Another source for natural Σ_2^p optimization problems is the class LOGSNP, defined by Papadimitriou and Yannakakis [64]. The definition of this class (and the closely related LOGNP) was motivated by the desire to precisely classify the complexity of certain problems solvable in $n^{O(\log n)}$ deterministic time. The class consists of all languages L expressible as:

$$L = \left\{ I : (\exists S \in [n]^{\log n})(\forall x \in [n]^p)(\exists j \in [\log n])\phi(I, s_j, x, j) \right\} \tag{5.10}$$

In [64], they identify several optimization problems complete for LOGSNP. We observe that the *succinct* versions of these problems lie in Σ_2^p – by blowing up the sizes of S, x, and j by an exponential, we obtain a problem expressible with the $\exists\forall$ quantifier alternation that seeks a polynomially large object S, such that for all x in an exponentially large set, some polynomial-time predicate involving I, S, and x is true. In contrast, the succinct versions of NP-complete problems tend to be NEXPTIME-complete.

In this section and the next, we show that the succinct versions of two of the more natural problems shown to be LOGSNP-complete in [64] are Σ_2^{p}-complete, and hard to approximate within very large factors. We begin with a succinct version of what Papadimitriou and Yannakakis call RICH HYPERGRAPH COVER:

SUCCINCT RICH HYPERGRAPH COVER:

INSTANCE: A *rich* hypergraph $G = (V, E)$ (one in which every hyperedge contains at least half of the vertices) represented succinctly by a circuit C_G: circuit $C_G(v, e) = 1$ if and only if vertex v is in hyperedge e.

SOLUTION: A node cover $E' \subseteq E$.

MEASURE: The size of cover E': $|E'|$.

Notice that in a *rich* hypergraph, there is always a node cover consisting of $\log |V|$ hyperedges; therefore, the decision version of this problem lies in Σ_2^{p}. In the reduction we see that the problem remains hard even when the encoding circuit is a DNF.

Theorem 5.9 *There is a polynomial-time gap-preserving reduction from* SUCCINCT SET COVER WITH GAP g *to* SUCCINCT RICH HYPERGRAPH COVER WITH GAP g *that maps instances of* SUCCINCT SET COVER *with m sets, and total DNF size s to instances of* SUCCINCT RICH HYPERGRAPH COVER *with circuit size $O(s \log m)$.*

Proof: We are given an instance of SUCCINCT SET COVER with m sets expressed as DNFs $\mathcal{C} = \{\phi_1, \phi_2, \ldots \phi_m\}$ over n variables. We may assume that every set covers at least half of the 2^n elements, by Remark 5.2. Our instance of SUCCINCT RICH HYPERGRAPH COVER is a circuit C_G encoding a hypergraph $G = (V, E)$ with 2^n vertices (identified with $\{0, 1\}^n$) and m hyperedges (identified with $[1 \ldots m]$). The circuit is defined as: $C(v, e) = \phi_e(v)$. Notice that C is expressible as a DNF with size $O(s \log m)$.

It is obvious that there is a set cover $\mathcal{C}' \subseteq \mathcal{C}$ of size k if and only if there is a node cover $E' \subseteq E$ of size k, since they are essentially the same object. ∎

5.9 Succinct tournament dominating set

We give a reduction for one more of the problems from [64], which seems to be the most natural of the LOGSNP-complete problems they identify (or at least the "log" limitation is the most obscured). It involves finding a dominating set in a tournament graph. Recall that a *tournament graph* is a directed graph in which exactly one of the two possible directed edges between each pair of vertices is present. A *dominating set* is a subset of the vertices so that every vertex is reachable in zero or one steps from a vertex in the subset.

SUCCINCT TOURNAMENT DOMINATING SET:

INSTANCE: A tournament graph $G(V, E)$, specified by a circuit C_G defined as follows: $C(x, y) = 1$ if and only if $(x, y) \in E$.

SOLUTION: A dominating set $V' \subseteq V$ for G.

MEASURE: The size of set V': $|V'|$.

Theorem 5.10 *There is a polynomial-time gap-preserving reduction from* SUCCINCT SET COVER WITH GAP g *to* SUCCINCT TOURNAMENT DOMINATING SET WITH GAP $\Omega(g)$ *that maps instances of* SUCCINCT SET COVER *with m sets over n variables and total DNF size s to instances of* SUCCINCT TOURNAMENT DOMINATING SET *with circuit size $O(mn)$.*

Proof: Given an instance of SUCCINCT SET COVER with m sets expressed as DNFs $\mathcal{C} = \{\phi_1, \phi_2, \dots \phi_m\}$ over n variables, we describe the graph $G(V, E)$. It will be clear that one can build the required circuit C_G efficiently, and that it has the stated size.

We use ideas from [64]. We need a construction due to [57] of a tournament graph on $p = 2^{O(m)}$ nodes with minimum dominating set size $m + 2$. We call this tournament graph T. Our graph G has m nodes $s_1, s_2, \dots s_m$, one for each set in the SUCCINCT SET COVER instance, plus an additional node s_0. In addition, we have $2^n + 1$ copies of T; we label these nodes t_{ij} for $i = 1, 2, \dots p$ and $j = 0, 1, \dots 2^n$ – node t_{ij} is the i-th node in the j-th copy of T. Intuitively, these subgraphs represent the 2^n elements in the ground set of the set-cover instance, plus one extra node.

We now describe how to connect the s and t nodes. First, we connect every t_{ij} node to s_0; then we add edge (s_ℓ, t_{ij}) for every i if $\phi_\ell(j) = 1$, and the opposite edge (t_{ij}, s_ℓ) for every i if $\phi_\ell(j) = 0$.

We connect $s_1, s_2, \ldots s_m$ in an arbitrary tournament, and add edge (s_0, s_ℓ) for $\ell = 1, 2, \ldots m$. Finally, we interconnect the t_{ij} nodes as follows:

- Within a copy: add edge $(t_{ij}, t_{i'j})$ if and only if edge (i, i') appears in T.
- Different nodes in different copies: add edge $(t_{ij}, t_{i'j'})$ if and only if edge (i, i') appears in T.
- Same nodes in different copies: add edge $(t_{ij}, t_{ij'})$ if and only if $j < j'$.

It is clear that G is now a tournament; we have considered every pair of vertices.

Claim 5.10.1 *There is a set cover $\mathcal{C}' \subseteq \mathcal{C}$ of size k if and only if G has a dominating set of size $k+1$, for $k \leq m$.*

Proof: ($\Rightarrow$) Given a cover $\mathcal{C}'$ of size k, we claim that the set $D = \{s_0\} \cup \{s_\ell | \phi_\ell \in \mathcal{C}'\}$ is a dominating set for G. Every s_ℓ, for $\ell > 0$ is dominated by s_0, and for every t_{ij}, there exists a $\phi_\ell \in \mathcal{C}'$ for which $\phi_\ell(j) = 1$ (because $\mathcal{C}'$ is a cover); therefore $s_\ell \in D$ dominates t_{ij} for all i. Set D has size $k+1$ as required.

($\Leftarrow$) Now, let D be a dominating set for G of size $k+1$. Node s_0 must be D, because if it is not, then the nodes of the 0-th copy of T must all be dominated by nodes in that copy of T, which requires $m+2 > |D|$ nodes, a contradiction.

Similarly, for $j = 1, 2, \ldots 2^n$, the j-th copy of T must be dominated by some node $s_\ell \in D$; otherwise all of the nodes of the j-th copy of T are dominated by nodes in the copies of T. The i-indices of these nodes form a dominating set for T itself, which must have size $m+2 > |D|$, a contradiction.

Therefore, for all $j \in \{0,1\}^n$, there exists an ℓ for which $\phi_\ell(j) = 1$; hence the set $\mathcal{C}' = \{\phi_\ell : s_\ell \in D\}$ is a set cover of size k. ■

This concludes the reduction from SUCCINCT SET COVER to SUCCINCT TOURNAMENT DOMINATING SET. ■

5.10 Additional maximization problems

We remark that several natural maximization variants of the problems we have considered so far seem to escape the complexity of Σ_2^{p}. A maximization version of SHORTEST

IMPLICANT CORE (different from MAXIMUM LITERAL DELETION) is LONGEST IMPLICANT EXTENSION, and it too is used as a substep in DNF minimization heuristics [23] (there it is called "reduce"):

LONGEST IMPLICANT EXTENSION:

INSTANCE: A DNF formula $\phi = t_1 \vee t_2 \vee \cdots \vee t_n$ consisting of n terms.

SOLUTION: An implicant $C' \supseteq t_n$ of ϕ.

MEASURE: The length of C': $|C'|$.

The decision version of this problem lies in the class $P_{||}^{\mathrm{NP}}$, "P with parallel access to NP," defined by Papadimitriou and Zachos [65]. This is true because conjunction C' can contain literal ℓ if and only if the conjunction $t_n\overline{\ell}$ is an implicant of $\phi - \{t_n\}$. We can therefore determine exactly which literals are permissible in C' with a polynomial number of (non-adaptive) queries to an NP oracle.

A second maximization problem, suggested by SUCCINCT SET COVER is the problem SUCCINCT SET PACKING, defined as follows:

SUCCINCT SET PACKING:

INSTANCE: A collection $\mathcal{C} = \{C_1, C_2, \ldots C_m\}$ of Boolean circuits over n variables.

SOLUTION: A subset $\mathcal{C}' \subseteq \mathcal{C}$ of disjoint sets; i.e., for all $C_i \neq C_j \in \mathcal{C}'$, there is no $x \in \{0,1\}^n$ for which $C_i(x) = C_j(x) = 1$.

MEASURE: The cardinality of the subset: $|\mathcal{C}'|$

This problem lies in P^{NP}, for the following reason: with polynomially many queries to an NP oracle, we can determine for all pairs of sets C_i, C_j, whether they are disjoint. Then, with one final (adaptive) NP query, we can solve the implied (non-succinct) set-packing problem.

5.11 Summary of problems and their inapproximability

The reductions in this chapter, together with Theorem 4.5 can be used to reach many different conclusions about the inapproximability of the problems. Some variables are:

problem	size measure N	ratio	assumption
SHORTEST IMPLICANT CORE	# literals		
SUCCINCT SET COVER	circuit size		
3-DNF SUCCINCT SET COVER	# literals	$N^{1/5-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{P}^{NP}$
MIN TERM DNF	# variables	$N^{1-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \tilde{P}^{NP}$
MIN DNF	# variables	$N^{1-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{ZPP}^{NP}$
IRREDUNDANT	# variables		
SUCCINCT RICH HYPERGRAPH COVER	circuit size		
SUCCINCT TOURNAMENT DOM. SET	circuit size	$N^{1/10-\epsilon}$ $N^{1/2-\epsilon}$ $N^{1/2-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{P}^{NP}$ $\Sigma_2^{\mathrm{p}} \not\subseteq \tilde{P}^{NP}$ $\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{ZPP}^{NP}$
MIN TERM DNF	# terms	$N^{1/15-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{P}^{NP}$
MIN DNF	# terms	$N^{1/3-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \tilde{P}^{NP}$
IRREDUNDANT	# terms	$N^{1/3-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{ZPP}^{NP}$
MIN TERM DNF	# literals	$N^{1/20-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{P}^{NP}$
MIN DNF	# literals	$N^{1/4-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \tilde{P}^{NP}$
IRREDUNDANT	# literals	$N^{1/4-\epsilon}$	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{ZPP}^{NP}$
MAXIMUM LITERAL DELETION	# literals		
MAXIMUM TERM DELETION	# literals	N^{δ}	$\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{P}^{NP}$

Table 5.1: Summary of inapproximability results for problems in this chapter. Here ϵ is any constant greater than zero, and δ is some constant greater than zero. $\tilde{P}$ denotes quasipolynomial time, i.e., $\mathrm{DTIME}(n^{O(\log n)})$.

the complexity of the algorithms we wish to rule out – this may range from P to $\mathrm{ZPP}^{\mathrm{NP}}$; the size measure of the problem at hand (usually the number of variables, terms or literal occurrences if the input is a DNF, and the number of variables or the circuit size if the input is a circuit); and the complexity assumption, which may be as strong as $\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{ZPP}^{\mathrm{NP}}$ or as weak as $\Sigma_2^{\mathrm{p}} \not\subseteq \mathrm{P}$. In Table 5.1, we give a sampling of some useful conclusions, which should all be read as: "there is no polynomial-time approximation algorithm for **problem** with NP oracle access achieving **ratio** unless **assumption** is false."

Note that for DNF formulae, # literals $\geq$ # terms $\geq$ # variables, so a hardness result in terms of the number of literals implies the same hardness for the other size measures. Similarly, for circuits, circuit size $\geq$ # variables, so a hardness result in terms of the circuit size implies one in terms of the number of variables.

Chapter 6

Shortest Implicant Problems

In this chapter, we examine a basic Σ_2^p optimization problem related to logic synthesis, namely the problem of finding shortest implicants. We consider three variants of the basic problem, in which the input is given as a DNF, a formula, and a circuit, respectively. The complexity and approximability of these problems (and the proof techniques) are dramatically different depending on the input representation type. Most surprisingly, the DNF version turns out to be much easier than anticipated: it can be solved using only $\log^2 n$-limited non-determinism and a coNP oracle, and indeed it is *complete* for the class of all problems solvable in this manner. We further show that the trivial approximation algorithm for this problem achieves a near-optimal approximation ratio. For the formula version, we obtain optimal (up to lower order terms) inapproximability results by giving a fairly delicate construction of formulae whose shortest implicants and implicates are almost as large as the formula themselves.

We need some preliminary definitions and a lemma. Given a Boolean function f, define $R_1(f)$ (resp. $R_0(f)$) to be the minimum number of variables that must be fixed (to zero or one) in order to make f identically 1 (resp. 0). Let $L(f)$ be the DeMorgan *formula complexity* of f; i.e., the size of the smallest formula over $\{\wedge, \vee, \neg\}$ that computes f.

Definition 6.1 *The* certificate complexity *of a Boolean function f is:*

$$R(f) \stackrel{def}{=} \min\{R_1(f), R_0(f)\}.$$

In other words, the certificate complexity of a f is the smallest number of variables that must be set to a constant to force the output to a constant. Given a Boolean function

f, we will often substitute another function for some of its inputs. The following lemma describes how this substitution alters $R_0(f)$ and $R_1(f)$.

Lemma 6.1 *Let g and $f(x_1, x_2, \dots x_n)$ be a Boolean function, and let $S \subseteq \{x_1, x_2, \dots x_n\}$ be the set of variables that will be substituted. Let $f' \stackrel{def}{=} f(h_1, h_2, \dots h_n)$ where h_i is an independent copy of g if $x_i \in S$, and $h_i \stackrel{def}{=} x_i$ otherwise. Then the following hold:*

1. *$R_1(f')$ (resp.$R_0(f)$) $\geq \min_C\{|vars(C) \setminus S| + R(g)|vars(C) \cap S|\}$, where C ranges over all implicants (resp. implicates) of f.*

2. *if C is an implicant (resp. implicate) of f with $vars(C) \cap S = \emptyset$, then C is an implicant (resp. implicate) of f'.*

Proof: The second claim is obvious; the first requires proof. Consider a shortest implicant C' of f'. For each i, either h_i is *forced* by C' to zero or one, or else h_i is *free* – it may take on both values in assignments consistent with C'. For each h_i that is forced, C' must include an implicant of h_i or $\neg h_i$, which requires $R(g)$ literals if $x_i \in W$, and one literal otherwise. There is also a corresponding implicant C of f that sets x_i to zero or one for each i for which h_i is forced. For this C, we have that $R(f') = |C'| \geq |\text{vars}(C) \setminus W| + R(g)|\text{vars}(C) \cap W|$. ■

6.1 Complexity of finding shortest implicants of a DNF

The problem we are concerned with in this section is the following:

SHORTEST IMPLICANT[DNF]:

INSTANCE: A DNF formula ϕ.

SOLUTION: An implicant C of ϕ.

MEASURE: The size of C: $|C|$.

The decision version of this problem exhibits the $\exists\,\forall$ quantifier alternation of Σ_2^{p} (it asks if there exists a short conjunction C such that for all x, $C(x) \rightarrow \phi(x)$), so we might expect it to be Σ_2^{p}-complete. It turns out, though, that the DNF input representation makes it a much easier problem than anticipated, due to the following combinatorial lemma:

Lemma 6.2 *Let $\phi = t_1 \vee t_2 \vee \ldots \vee t_n$ be a DNF formula. Then any shortest implicant C of ϕ may be obtained from some term t_i with at most $\log n$ additions of literals and at most $\log n$ deletions of literals.*

Proof: Let C be a shortest implicant of ϕ, and let us rearrange the terms of ϕ so that $t_1, t_2, \ldots, t_{n'}$ are exactly those terms of ϕ that are consistent with C. Now, for all i, define $t'_i = t_i \wedge C$; this ensures that $C \subseteq t'_i$ for all i. We first argue that there must exist an i for which $|t'_i| \leq |C| + \log n'$. Suppose there were not. Then each t'_i has length greater than $|C| + \log n'$, and the maximum number of assignments covered by the terms $t'_1, t'_2, \ldots t'_{n'}$ is therefore strictly less than $n' 2^{m-|C|-\log n'} = 2^{m-|C|}$, where m is the number of variables in ϕ. This is a contradiction, since it implies that the terms consistent with C do not cover all of the assignments covered by C. So there must be an i for which $|t'_i| \leq |C| + \log n'$.

Conjunction t'_i was obtained from t_i by adding literals. We cannot have added more than $\log n'$ literals, unless $|t_i| < |C|$, which contradicts C being a shortest implicant of ϕ. Since $n' \leq n$, C is obtainable from t_i with at most $\log n$ insertions (to get t'_i) and at most $\log n$ deletions of literals (to recover C). ■

As a consequence of Lemma 6.2, the decision version of SHORTEST IMPLICANT[DNF] requires only $O(\log^2)$-limited non-determinism and a coNP oracle. Given an instance $\langle \phi = t_1 \vee t_2 \vee \ldots \vee t_n, k \rangle$, we can guess an index i, a set I of $\log n$ literals to add, a set D of $\log n$ literals to delete, and construct a candidate conjunction $C = (t_i \cup I) \setminus D$. We accept if C is well-formed, has length at most k, and a query to the coNP oracle indicates that it is an implicant of ϕ.

It is natural to ask whether perhaps a stronger lemma is true that might allow us to solve the decision version of SHORTEST IMPLICANT[DNF] in coNP. We give evidence that this problem is not in coNP, by showing that it is *complete* for a complexity class encompassing all problems that can be solved using $O(\log^2 n)$-limited non-determinism and a coNP oracle. We call this class GC($\log^2 n$, coNP) after the "guess then check" framework of Cai and Chen [16] (however they are predominantly concerned with classes in this framework with far weaker verifiers); its definition follows:

Definition 6.2 *The complexity class* GC($\log^2 n$, coNP) *is the class of all languages L expressible as:*

$$L = \{I : \exists X \textit{ such that } R(I, X)\},$$

where $|X| \leq O(\log^2 |I|)$ *and* R *is a coNP predicate.*

Notice that the usual characterization of Σ_2^p is identical except that $|X|$ is bounded only by some polynomial in $|I|$.

It is straightforward to see that the class $\mathrm{GC}(\log^2 n, \mathrm{coNP})$ lies between coNP and Σ_2^p in the polynomial hierarchy. It seems unlikely that $\mathrm{GC}(\log^2 n, \mathrm{coNP})$ contains NP, although to our knowledge, formal evidence in support of this conjecture is not known. More on related complexity classes can be found in [16]. Limited non-determinism in the first level of the hierarchy has been studied by a number of authors [43, 24, 64]. We now turn to the proof that the decision version of SHORTEST IMPLICANT[DNF] is $\mathrm{GC}(\log^2 n, \mathrm{coNP})$-complete.

Theorem 6.3 *The decision version of* SHORTEST IMPLICANT[DNF] *is* $GC(\log^2 n, coNP)$*-complete.*

Proof: Membership in the class follows from Lemma 6.2. Hardness is via a generic reduction. Let L be a language in $\mathrm{GC}(\log^2 n, \mathrm{coNP})$, let I be an instance, and let $R(I, X)$ be the coNP predicate guaranteed by the definition of $\mathrm{GC}(\log^2 n, \mathrm{coNP})$. Let $k = |X|$, so $k = c \log^2 n$ for some constant c.

From here, the proof is similar to the proof of Theorem 5.1. Define the set $S = \{s_{ij} | 1 \leq i \leq c \log n, 1 \leq j \leq n\}$. Consider the coNP algorithm A whose input is a $cn \log n$-bit vector that is interpreted as representing a subset $S' \subseteq S$. If $|S'| \geq c \log n + 1$, then A accepts. Otherwise, if $|S'| = c \log n$ and for all i, there exists a unique j for which $s_{ij} \in S'$, then A constructs an assignment to the X variables as follows: partition $x_1, x_2, \ldots x_{c \log^2 n}$ into $c \log n$ groups of $\log n$ variables each (i.e., group 1 contains $x_1, x_2, \ldots x_{\log n}$; group 2 contains $x_{\log n+1}, x_{\log n+2}, \ldots x_{2 \log n}$; etc...); for each i, if j is the unique index for which $s_{ij} \in S'$, assign the i-th group the j-th possible assignment (there are exactly n possible assignments to each group). Now, A accepts if $R(I, X)$ accepts, and rejects otherwise.

Notice that A accepts a non-empty monotone set: if accepts all strings of weight at least $c \log n + 1$, and possibly some of the strings of weight $c \log n$. Furthermore, A accepts a string with weight $c \log n$ if and only if $I \in L$.

Now by Lemma 1.4 and Lemma 1.3, we can construct a 3-DNF $\phi(S, Y)$ such that $\forall Y \; \phi(s, Y) = 1$ if and only if A accepts s. We construct DNF ϕ' from ϕ be substituting parity on $c \log n + 1$ variables for each of the Y variables. Notice that the resulting formula is still only polynomially large when multiplied out into DNF, because the DNF representations of the parity formulae each have $2^{c\log n+1}/2 = n^c$ terms, and there are at most a constant number of substitutions per term of ϕ.

Claim 6.3.1 *Formula ϕ' has an implicant of length at most $c\log n$ if and only if $I \in L$.*

Proof: ($\Rightarrow$) If there exists an implicant of ϕ' of length at most $c\log n$, then there must exist an implicant C of ϕ of length at most $c\log n$ that consists of only the S variables. If not, then the shortest implicant of ϕ' has length at least $c\log n + 1$ by Lemma 6.1. Therefore, if z is an assignment to the S variables with z_i set to 1 if and only if $z_i \in C$, then we have $\forall Y\ \phi(z,Y)$, which implies A accepts z, and because z has weight $c\log n$, this implies that $I \in L$.

($\Leftarrow$) If $I \in L$, then A accepts a string z of weight $c\log n$, and so $\forall Y\ \phi(z,Y) = 1$. By monotonicity, the conjunction C of the variables consisting of the 1's in z is an implicant of ϕ; as C contains no Y variables, it is also an implicant of ϕ', by Lemma 6.1. Therefore there is an implicant of ϕ' of length $c\log n$. ■

This concludes the proof that the decision version of SHORTEST IMPLICANT[DNF] is $\mathrm{GC}(\log^2 n, \mathrm{coNP})$-complete. ■

The complement of $\mathrm{GC}(\log^2 n, \mathrm{coNP})$ also contains some interesting natural problems, notably:

ROBUST CNF-SAT:

INSTANCE: A CNF formula ϕ and an integer k.

QUESTION: Is ϕ_ρ satisfiable for all restrictions ρ that set at most k variables to constants?

which is simply the complement of (the decision version of) the CNF version of SHORTEST IMPLICANT[DNF] (which asks for the shortest implic*ate* of a CNF formula). By the previous theorem, it is complete for the complement of $\mathrm{GC}(\log^2 n, \mathrm{coNP})$. The "robust" versions of some other NP-complete problems are also complete for this class, although the $\log n$ restriction appears explicitly in the problem statement. For example, one can show that LOG-ROBUST DOMINATING SET (the problem of determining whether a graph has a small dominating set even after the deletion of at most $\log n$ edges) is complete for the complement of $\mathrm{GC}(\log^2 n, \mathrm{coNP})$.

6.2 Approximability of shortest implicant[dnf]

By Lemma 6.2 in the previous section, the algorithm that returns the shortest term in the input DNF formula is an approximation algorithm that is guaranteed to return an implicant that is within a $\log m$ *additive* factor of optimal, where m is the number of terms in the input formula. In the next theorem we show that this (trivial) algorithm is optimal up to a small constant factor. We need the following simple lemma:

Lemma 6.4 *If $\sum_{i=1}^{k} a_i \leq B$, where the a_i are non-negative real numbers, then $\prod_{i=1}^{k} a_i \leq (B/k)^k$.*

Proof: WLOG, assume $\sum_{i=1}^{k} a_i = B$. The lemma follows from the observation that $\prod_{i=1}^{k} a_i$ achieves its maximum when the a_i are all equal; i.e., $a_i = B/k$ for all i. ∎

Theorem 6.5 SHORTEST IMPLICANT[DNF] *is $GC(\log^2 n, coNP)$-hard to approximate to within a $(1/3-\epsilon)\log m$ additive factor, where m is the number of terms in the input formula, for any constant $\epsilon > 0$.*

Proof: The proof may be seen as a generalization of the proof of Theorem 6.3, the difference being that here we introduce an additive gap between the optima in the positive and negative cases.

As before, let L be a language in $\mathrm{GC}(\log^2 n, \mathrm{coNP})$, let I be an instance, and let $R(I, X)$ be the coNP predicate guaranteed by the definition of $\mathrm{GC}(\log^2 n, \mathrm{coNP})$. Let $k = |X| = c\log^2 n$, for some constant c. Define the set $S = \{s_{ij} | 1 \leq i \leq c\log n, 1 \leq j \leq n\}$, and consider the coNP algorithm A whose input is a $cn\log n$-bit vector that is interpreted as representing a subset $S' \subseteq S$.

The proofs diverge here. Let $c' > c$ be a constant that will depend on ϵ. If $|S'| \geq c'\log n + 1$, then A accepts. Otherwise, check if for all i, there exists some j for which $s_{ij} \in S'$. If not, reject; otherwise, proceed.

Define a mapping σ from subsets $S'' \subseteq S$ that include a unique element s_{ij} for each i to assignments X to the variables $x_1, x_2, \ldots x_{c\log^2 n}$. As before, we partition $x_1, x_2, \ldots x_{c\log^2 n}$ into $c\log n$ groups of $\log n$ variables each; then, for each i, if j is the unique index for which $s_{ij} \in S''$, we assign the i-th group the j-th possible assignment.

Define $S_i = \{s_{ij} | s_{ij} \in S'\}$. For every set S'' formed by choosing exactly one element from S_i for all i, check if $R(I, \sigma(S'')$ accepts. By Lemma 6.4, we need to check at most

$(c' \log n / c \log n)^{c \log n} = n^{c \log_2 c'/c}$ assignments, and by Lemma 4.1 this can be done in coNP. Finally, we accept if $R(I, \sigma(S''))$ accepts for any of the assignments checked.

Claim 6.5.1 *CoNP algorithm A accepts exactly a non-empty monotone set.*

Proof: Let $S' \subseteq S$ be an input accepted by A, and let S'' be any superset of S'. Then either $|S''| \geq c' \log n + 1$, and A accepts, or else there must be some assignment X encoded by S' that causes $R(I, X)$ to accept; it is easy to verify that algorithm A checks the same assignment on input S'', and therefore accepts. Finally, notice that A accepts the all-ones string. ■

Notice also that if $I \in L$, then A accepts a string with weight $c \log n$; if $I \notin L$, then the smallest weight string accepted by A has weight $c' \log n + 1$.

We now consider the running time of A. After determining the weight of the input, A computes at most $n^{c \log_2 c'/c}$ assignments and evaluates a coNP predicate on them. This requires time $q(n, c') = n^{\alpha c \log_2 (c'/c)}$ for some constant α. By Lemma 1.4, we can build a Π_1 ND circuit C from A of size $O(q(n, c')^2)$. By Lemma 1.3, we can build a 3-DNF $\phi(S, Y)$ such that $\forall Y \; \phi(s, Y) = 1$ if and only if C accepts s. Formula ϕ has at most $O(q(n, c')^2)$ terms.

We construct DNF ϕ' from ϕ as follows: for every Y variable, we substitute parity on $c' \log n + 1$ variables. This increases the number of terms by a factor of at most $n^{3c'}$, as each parity formula has $2^{c' \log n + 1}/2 = n^{c'}$ terms and there are at most one three substitutions per term. Therefore, ϕ'' has at most $m = O(q(n, c')^2) \cdot n^{3c'}$ terms.

Claim 6.5.2 *If $I \in L$, ϕ' has an implicant of length at most $c \log n$.*

Proof: If $I \in L$, then A accepts a string w of weight $c \log n$, and so $\forall Y \; \phi(w, Y) = 1$. By monotonicity, the conjunction C of the variables consisting of the ones in w is an implicant of ϕ. Since C contains no Y variables, by Lemma 6.1, C is also an implicant of ϕ', and it has length $c \log n$ as required. ■

Claim 6.5.3 *If $I \notin L$, then the smallest implicant of ϕ' has length at least $c' \log n + 1$.*

Proof: We prove the contrapositive. If there exists an implicant of ϕ' of length at most $c' \log n$, then there must exist an implicant C of ϕ of length at most $c' \log n$ that excludes the Y variables. If not, then the shortest implicant of ϕ' has length at least $c' \log n + 1$ by

Lemma 6.1. Now, if w is an minimum weight assignment to the S variables consistent with C, then we have $(\forall Y)\phi(w, Y)$; and hence algorithm A accepts w. Since w has weight at most $c' \log n$, it must be that $I \in L$. ■

Now, DNF ϕ' has $m = n^{3c'+O(\log c')}$ terms, and the additive gap is $(c' - c) \log n$. If we choose $c' = \Omega(\epsilon^{-1}c)$, then for sufficiently large n:

$$(c' - c) \log_2 n \geq (1/3 - \epsilon) \log_2 m,$$

as required. ■

6.3 The formula and circuit versions

We turn now to the formula and circuit versions of the shortest implicant problem:

SHORTEST IMPLICANT[FORMULA]:

INSTANCE: A DeMorgan formula ϕ.

SOLUTION: An implicant C of ϕ.

MEASURE: The size of C: $|C|$.

SHORTEST IMPLICANT[CIRCUIT]:

INSTANCE: A circuit ϕ.

SOLUTION: An implicant C of ϕ.

MEASURE: The size of C: $|C|$.

We begin with a simple proof that the two problems are Σ_2^p-complete.

Theorem 6.6 *The decision versions of* SHORTEST IMPLICANT[FORMULA] *and* SHORTEST IMPLICANT[CIRCUIT] *are Σ_2^p-complete.*

Proof: We give the reduction for SHORTEST IMPLICANT[FORMULA], which is a special case of SHORTEST IMPLICANT[CIRCUIT].

Let $\phi(X_1, X_2)$ be an instance of QSAT$_2$, and let $X_1 = \{x_1, x_2, \ldots x_m\}$. This instance is a positive instance if and only if there exists an implicant of ϕ consisting of only variables in X_1. We modify ϕ so that *all* implicants must include the variables in X_1 and m additional variables. In the resulting formula, an implicant consisting of *exactly* these $2m$ variables exists if and only if there is an implicant of ϕ consisting of only variables in X_1.

We introduce new variables $w_1, w_2 \ldots w_m$ and define ϕ' as follows:

$$\begin{aligned} \phi' &= \phi \wedge \bigwedge_{i=1}^{m} (x_i \oplus w_i) \\ &= \phi \wedge \bigwedge_{i=1}^{m} (x_i \overline{w_i} \vee \overline{x_i} w_i) \end{aligned}$$

We claim that ϕ' has an implicant of length at most $2m$ if and only if the instance of QSAT$_2$ is a positive instance.

($\Rightarrow$) Let C' be an implicant of ϕ'. Clearly vars(C') contains $x_1, x_2, \ldots x_m$ and $w_1, w_2, \ldots w_m$; otherwise there is an assignment consistent with C' in which $(x_i \oplus w_i)$ is false for some i. Because $|C'| \leq 2m$, vars(C') equals *exactly* $X_1 \cup \{w_1, w_2, \ldots w_m\}$. Since ϕ' is the conjunction of ϕ with additional formulae, C' is also an implicant of ϕ, which implies that there exists an implicant of ϕ consisting of exactly the variables in X_1 (simply restrict C' to the variables that appear in ϕ). Hence the instance of QSAT$_2$ is a positive instance.

($\Leftarrow$) If the QSAT$_2$ instance is a positive instance, then there is an implicant C of ϕ consisting of exactly the variables in X_1. Construct conjunction C' by augmenting C with w_i literals as follows: if C includes literal x_i, add $\overline{w_i}$; if C includes literal $\overline{x_i}$, add literal w_i. Conjunction C' is an implicant of ϕ' of length $2m$ as required. ■

We now proceed to showing the hardness of approximating SHORTEST IMPLICANT[FORMULA] and SHORTEST IMPLICANT[CIRCUIT]. The following observation is critical: in the proof of Theorem 6.6, it happens that *all* implicants of ϕ' contain at least one variable from X_2 if and only if the instance is a negative instance. In the following proofs, we use parity substitution to "weight" the variables in X_2. The result is that all implicants (and hence, all shortest implicants) of a negative instance are long, while a positive instance has a short implicant that avoids the weighted variables.

We will also use the following facts about parity formulae and circuits over $(\wedge, \vee, \neg)$ (see [87]): the parity of n variables can be computed by a circuit with $O(n)$ gates, and by a formula with size $O(n^2)$.

In the next theorem, the value n in the approximation factor is the size of the formula (number of leaves).

Theorem 6.7 *Approximating* SHORTEST IMPLICANT[FORMULA] *to within an* $n^{1/2-\epsilon}$ *factor is* Σ_2^p*-hard, for any constant* $\epsilon > 0$.

Proof: Recall from the proof of Theorem 6.6 that we transformed an instance of QSAT$_2$ given by $\phi(X_1, X_2)$ into the following formula:

$$\phi' = \phi \wedge \bigwedge_{i=1}^{m} (x_i\overline{w_i} \vee \overline{x_i}w_i) \,,$$

where $x_1, x_2, \ldots x_m$ are the variables in X_1. Let $y_1, y_2, \ldots y_p$ be the variables in X_2, and let $N = |\phi'|$. Let ϕ'' be the "weighted" version of ϕ', obtained by substituting, for each y_i, a parity formula on N^c new variables. Here, c is a constant that will depend on ϵ. The size of each substituted parity formula is $O(N^{2c})$, so $|\phi''| \leq \alpha N^{2c+1}$, where α is the constant in the big-oh notation.

As observed above, if the instance of QSAT$_2$ is a positive instance, then there exists an implicant C of ϕ' of length $2m$, consisting of exactly the variables $x_1, x_2, \ldots x_m$ and $w_1, w_2, \ldots w_m$. By Lemma 6.1, C is also an implicant of ϕ''. However, if the instance of QSAT$_2$ is a negative instance, then every implicant of ϕ' includes at least one of the y variables. Again by Lemma 6.1, this implies that $R_1(\phi'') \geq N^c$.

Using the fact that $2m \leq N$, we have the following inequality for sufficiently large N and $c \geq (3/4)\epsilon^{-1}$:

$$\frac{N^c}{2m} > \left(\alpha N^{(2c+1)}\right)^{(1/2-\epsilon)} \,,$$

which establishes the required gap. ∎

Using the same method, we obtain an even stronger result for the version of the problem in which the input is a circuit. Here, the value n in the approximation factor is the size of the circuit (number of gates).

Theorem 6.8 *Approximating* SHORTEST IMPLICANT[CIRCUIT] *to within an* $n^{1-\epsilon}$ *factor is* Σ_2^p*-hard, for any constant* $\epsilon > 0$.

Proof: Analogous to the previous proof. We start with the same formula ϕ' but obtain *circuit* ϕ'' by substituting parity *circuits* on N^c variables for each y_i. The size of ϕ'' is at most αN^{c+1}. The same analysis as used above regarding the sizes of shortest implicants in the positive and negative cases holds.

The following inequality holds for sufficiently large N, and $c \geq 2\epsilon^{-1}$:

$$\frac{N^c}{2m} > \left(\alpha N^{(c+1)}\right)^{(1-\epsilon)}.$$

This establishes the required gap. ■

Notice that the 1/2 in the approximation factor in Theorem 6.7 comes from the fact that the parity formulae have quadratic size. By Krapchenko's lower bound (cf. [87]), smaller parity formulae are not possible. However, there may exist smaller formulae that retain the critical property of the parity formulae used in the proof: namely, they have no "short" implicants or implicates. The ratio we are interested in is $E(f) = \log L(f)/\log R(f)$.

If there exists an efficiently constructible formula family $F = (f_i)$ for which $E(f_i) \leq r$ for all i, then we can use that family in place of the parity formulae in the above proof to obtain improved inapproximability to within an $n^{1/r-\epsilon}$ factor. We examine this question in depth in the next section, and we succeed in constructing a uniform family $F = (f_i)$ of Boolean functions with $E(f_i)$ arbitrarily close to 1 for all i (see Theorem 6.11). From this we have the following improved inapproximability result for SHORTEST IMPLICANT[FORMULA]:

Theorem 6.9 SHORTEST IMPLICANT[FORMULA] *is Σ_2^p-hard to approximate to within an $n^{1-\epsilon}$ factor, for any constant $\epsilon > 0$.*

Proof: Construct an f_i for which $R_1(f_i)$ and $R_0(f_i)$ are at least the desired length, and substitute it as we did with the parity formula in the proof of Theorem 6.7. ■

6.4 Small formulae with large certificate complexity

We note that our $R(f)$ is the same as a quantity called $\ell_{\min}(f)$, investigated by Bublitz, Schürfeld, and Wegener in another context [15]. They are interested in its relationship to various other complexity measures, while we are interested in its relationship to formula size. We wish to construct a uniform family of Boolean functions $f = \{f_i\}$ with small formula complexity $L(f_i)$, but with $R(f_i)$ as large as possible. This motivates the following definition:

Definition 6.3 *The* certificate complexity exponent *of a Boolean function f is:*

$$E(f) \stackrel{def}{=} \frac{\log L(f)}{\log R(f)}.$$

We observe that $E(f) \geq 1$ for all Boolean functions f, and that $E(\text{PARITY}) = 2$ (because we know that $L(\text{PARITY}_n) = \Theta(n^2)$ [87]). In the following lemma, we prove that $E(f)$ behaves well under function composition, which reduces the problem of finding a uniform family of Boolean functions with small nondeterministic certificate complexity exponents to the problem of showing the *existence* of a single such function f.

Lemma 6.10 (Composition Lemma) *Let $f(x_1, x_2, \ldots, x_n)$ be a Boolean function, and let $f' = f(f_1, f_2, \ldots, f_n)$, where $f_1, f_2, \ldots, f_n$ are independent copies of f. Then $E(f') \leq E(f)$.*

Proof: By Lemma 6.1, we see that $R_0(f') \geq R(f)R_0(f)$ and $R_1(f') \geq R(f)R_1(f)$; therefore $R(f') \geq R(f)^2$. Also, $L(f') \leq L(f)^2$. Therefore, as required:

$$E(f') = \frac{\log L(f')}{\log R(f')} \leq \frac{2\log L(f)}{2\log R(f)} = E(f).$$

■

The remainder of this section is devoted to proving the following theorem:

Theorem 6.11 *For every constant $\delta > 0$, there exists a Boolean function f for which $E(f) \leq 1 + \delta$.*

from which we can deduce the existence of the a uniform family of functions with the same property:

Theorem 6.12 *For every constant $\delta > 0$, there exists a family of Boolean functions $f = \{f_i\}$ for which $E(f_i) \leq 1 + \delta$ for all i.*

Proof: We describe the algorithm to construct formulae for such functions. First, the algorithm exhaustively searches for the lexicographically first DeMorgan formula ϕ computing a function f for which $E(f) \leq 1 + \delta$. Call this formula f_0. We obtain f_i for $i \geq 1$ by composing f_0 with itself i times. ■

6.4.1 Lower bounds on $E(f)$

Before proving Theorem 6.11, we note that relatively simple formula constructions will not suffice for proving the theorem.

Theorem 6.13 *Let f be a monotone or read-once formula. Then $|f| \geq R(f)^2$.*

Proof: We show that $|f| \geq R_0(f)R_1(f)$, by induction on the structure of formula f. As the base case, if f is the single variable x, then $1 = |f| \geq R_0(f)R_1(f) = 1$. If $f = \neg g$, for some formula g, then by induction $|g| \geq R_0(g)R_1(g)$, and clearly $|f| > |g| \geq R_0(g)R_1(g) = R_0(f)R_1(f)$, since $R_0(f) = R_1(\neg f)$ and $R_1(f) = R_0(\neg f)$.

Now consider the case in which $f = g \wedge h$, for two formulas g and h. First, notice that $R_0(g \wedge h) \leq \min\{R_0(g), R_0(h)\}$. Let C_g be a shortest implicant of g and let C_h be a shortest implicant of h. If f is a monotone formula, then g and h are monotone formulas, and then neither implicant includes a negative literal. Therefore C_g and C_h are consistent with each other. Consistency also holds if f is a read-once formula, because then $\mathrm{vars}(C_g) \cap \mathrm{vars}(C_h) = \emptyset$. In either case, $C_g \wedge C_h$ is an implicant of $g \wedge h$ and therefore $R_1(g \wedge h) \leq |S_g| + |S_h| = R_1(g) + R_1(h)$. Without loss of generality, assume $R_0(g) \leq R_0(h)$; then we have:

$$\begin{aligned} |f| &= |g| + |h| \geq R_0(g)R_1(g) + R_0(h)R_1(h) \\ &\geq R_0(g)[R_1(g) + R_1(h)] \geq R_0(f)R_1(f) \end{aligned}$$

The argument for the final case in which $f = g \vee h$ is essentially the same. The stated lower bound follows from the observation that $R(f)^2 = \min\{R_0(f), R_1(f)\}^2 \leq R_0(f)R_1(f) \leq |f|$. ■

6.4.2 Proof of Theorem 6.11

We now proceed with the proof of Theorem 6.11. The following is the key lemma:

Lemma 6.14 *If there exists a formula g with $E(g) = c$ then there exists a formula g' with $E(g') \leq 2 - 1/c + \epsilon$ for any $\epsilon > 0$.*

Proof: First, we use the Composition Lemma to obtain a sequence of formulae $g_1, g_2, g_3, \ldots$ for which $E(g_i) \leq c$ for all i: let g_1 be g composed with itself so that $a = R(g_1)$ is sufficiently large (this requirement will be made more precise below), and define g_n for $n > 1$ to be g_1 composed with itself $n-1$ times, so $R(g_n) \geq a^n$. Let m be the smallest even integer such that $m \geq a^c$.

Next, define recursively a sequence of formulae $f_0, f_1, f_2, \ldots$ built from $g_1, g_2, \ldots$. In the following, superscripts identify multiple independent copies of each formula, and "maj" is a formula that is true if at least $m/2$ of its m arguments are true, where m is even.

$$f_0 = x \qquad f_n = \bigwedge_{i=1}^{m} \left(g_n^{(i)} \neg \mathrm{maj}(g_n^{(1)}, \ldots g_n^{(m)}) \vee f_{n-1}^{(i)} \right)$$

For $n > 0$, f_n is true iff for at least $(m/2+1)$ distinct values of i, $g_n^{(i)}$ is zero and $f_{n-1}^{(i)}$ is one, and for all other values of i, either $g_n^{(i)}$ is one or $f_{n-1}^{(i)}$ is one. In the next three claims we determine bounds on $R_0(f_n)$, $R_1(f_n)$ and $L(f_n)$.

Claim 6.14.1 *If $m/2 \geq a$, then $R_0(f_n) \geq a^n$ for all n.*

Proof: An implicate of f_n $(n > 0)$ must either (1) force $g_n^{(i)}$ to zero and $f_{n-1}^{(i)}$ to zero for some i, or (2) for at least $m/2$ values of i force $g_n^{(i)}$ to one or $f_{n-1}^{(i)}$ to zero. In the first case, the implicate has length at least $a^n + R_0(f_{n-1})$; in the second, the implicate has length at least $(m/2)\min\{a^n, R_0(f_{n-1})\}$. We prove the claim by induction on n. Clearly $R_0(f_0) = 1$; now assume $R_0(f_{n-1}) \geq a^{n-1}$. In the first case, the claim holds trivially; in the second case, $R_0(f_n) \geq (m/2)\min\{a^n, R_0(f_{n-1})\} \geq a(a^{n-1}) = a^n$, using the fact that $m/2 \geq a$ and the induction hypothesis. ■

Claim 6.14.2 *For all $n > 0$, $(m/2)^n \leq R_1(f_n) \leq m^n$.*

Proof: An implicant of f_n $(n > 0)$ must force at least $m/2+1$ of the f_{n-1} to one. By induction on n, we conclude that $R_1(f_n) \geq (m/2+1)^n \geq (m/2)^n$.

For the upper bound, notice that forcing all of the f_{n-1} to one forces the whole formula to one, so for $n > 0$, $R_1(f_n) \leq mR_1(f_{n-1})$. Since $R_1(f_0) = 1$, we have $R_1(f_n) \leq m^n$, as required. ■

Claim 6.14.3 *There exists a constant d, such that for all n, $L(f_n) = O(nm^{n+d})$.*

Proof: We have the following recurrence: $L(f_0) = 1$ and $L(f_n) \leq p(m)a^{cn} + mL(f_{n-1}) \leq p(m)m^n + mL(f_{n-1})$, where $p(m) \leq O(m^d)$ is a fixed polynomial in m accounting for all of the occurrences of copies of g_n. Our induction hypothesis is that $L(f_n) \leq m^n + p(m)nm^n$. Clearly this holds for $n = 0$. We have:

$$\begin{aligned} L(f_n) &\leq p(m)m^n + mL(f_{n-1}) \leq p(m)m^n + m[m^{n-1} + p(m)(n-1)m^{n-1}] \\ &\leq p(m)m^n + m[p(m)nm^{n-1}] = p(m)m^n + p(m)nm^n. \end{aligned}$$

Finally, note that $m^n + p(m)nm^n = O(nm^{n+d})$. ■

Now, we take g'_n to be the disjunction of $k = \lceil R_1(f_n)/R_0(f_n) \rceil$ independent copies of f_n. This has the effect of multiplying the length of the shortest implicate by k, so that $R(g'_n) = R_1(f_n)$. We have:

$$R(g'_n) \leq \frac{\log\left(O(nm^{n+d})\lceil \frac{m^n}{a^n} \rceil\right)}{\log((m/2)^n)} \leq \frac{\log\left(O(nm^d(m^2/a)^n)\right)}{\log((m/2)^n)}$$

As n approaches infinity, this last expression approaches $\log(m^2/a)/\log(m/2)$. By choosing a sufficiently large initially, this value can be made arbitrarily close to $2 - 1/c$. Then taking n sufficiently large yields a formula $g' = g'_n$ with $R(g') \leq 2 - 1/c + \epsilon$ for any $\epsilon > 0$.

One detail remains: the final step in the construction makes sense only if $R_1(f_n) \geq R_0(f_n)$; if this inequality does not hold, then we observe that f_n itself satisfies the lemma. ■

Now we apply this lemma repeatedly, starting with parity (for which $c = 2$), to show that for any $\delta > 0$, there exists an f with $R(f) \leq 1 + \delta$, thus proving Theorem 6.11.

Chapter 7

Approximation Algorithms for Minimization Problems

In this chapter we describe approximation algorithms for many of the minimization problems considered in this thesis. Since these problems are Σ_k^{P}-complete, our approximation algorithms will use a $\Sigma_{k-1}^{\mathrm{P}}$ oracle. The algorithms run in randomized polynomial time, with the interesting feature that approximation quality and (asymptotic) running time can be traded for each other.

These algorithms are based on results in [14] concerning exact learning using equivalence queries. Some of the ideas also appear in [50]. We also use a result of Bellare, Goldreich, and Petrank [8] (which is a strengthening of a classic result of Jerrum, Valiant, and Vazirani [37]) that allows us to sample *uniformly* from an efficiently recognizable set, using an oracle:

Theorem 7.1 ([8]) *Let $S \subseteq \{0,1\}^n$ be a subset whose characteristic function can be evaluated in $DTIME(T(n))^{\Sigma_{k-1}^P}$. Then there is a randomized procedure running in time* $\mathrm{poly}(T(n))$ *with access to a Σ_k^P oracle that "fails" with probability at most* $1/2$, *and, conditioned on not failing, outputs a uniformly distributed $x \in S$.*

Actually, we have generalized the theorem from [8] somewhat; they state it for $k = 1$, and we notice here that their procedure relativizes. By repeating this procedure t times, and outputting the first x returned in a non-failing run (if there is one), we can lower the failure probability to 2^{-t}.

7.1 The generic approximation algorithm

We begin by describing a very general framework that yields approximation algorithms for many of our problems. This framework potentially applies to any Σ_k^{P} minimization problem with $k \geq 2$. There are several ingredients that we need to specialize this framework for a particular minimization problem:

Problem: A Σ_k^{P} minimization problem $A = (I_A, \mathrm{sol}_A, m_A, \min)$, an efficiently computable upper bound function $B(x)$ such that $B(x) \geq \mathrm{opt}_A(x)$, and a $\Sigma_{k-2}^{\mathrm{P}}$ predicate Q with the property that for all instances $x \in I_A$,

$$(\forall z) Q(x, y, z) \Leftrightarrow y \in \mathrm{sol}_A(x). \tag{7.1}$$

Some such predicate exists because by the definition of Σ_k^{P} optimization problems, the set sol_A is recognizable by a Π_{k-1}^{P} predicate. Let $p_y(|x|)$ and $p_z(|x|)$ be (polynomial) functions bounding the length of y and z in terms of the size of instance x.

Base sets: For all $1 \leq K \leq B(x)$, a set $\textsc{base}(x, K) \subseteq \{0,1\}^{p_y(|x|)}$ with the following properties:

1. $\{y \in \mathrm{sol}_A(x) : m_A(x, y) \leq K\} \subseteq \textsc{base}(x, K)$, and
2. $\textsc{base}(x, K)$ is recognizable by a $\Sigma_{k-2}^{\mathrm{P}}$ predicate.

(α, β)**-merger:** A polynomial-time computable function

$$\textsc{merge} : \textsc{base}(x, K)^{\beta} \to \{0,1\}^{p_y(|x|)}$$

with the following properties:

1. for all $y_1, y_2, \ldots y_\beta \in \textsc{base}(x, K)$,

$$\textsc{merge}(y_1, y_2, \ldots y_\beta) \in \mathrm{sol}_A(x) \Rightarrow m_A(x, \textsc{merge}(y_1, y_2, \ldots y_\beta)) \leq \beta K,$$

2. for all subsets $S \subseteq \textsc{base}(x, K)$ and $z \in \{0,1\}^{p_z(|x|)}$,

$$\Pr_{y \in S}[Q(x, y, z) = 0] \leq \alpha \Rightarrow$$
$$\Pr_{y_1, \ldots y_\beta \in S}[Q(x, \textsc{merge}(y_1, y_2, \ldots y_\beta), z) = 0] \leq \frac{1}{2^{2p_z(|x|)+1} \log B(x)}.$$

Given these ingredients, we can define the generic approximation algorithm. In the following description, the sets S_i are stored implicitly – we will be able to check membership and emptiness using an $\Sigma_{k-1}^{\mathrm{P}}$ oracle.

ALGORITHM GENERIC_APPROX(x)

perform binary search to find the smallest $K \in [1, B(x)]$ for which FIND_SOLUTION(x, K) returns (y, "yes"); output y.

PROC FIND_SOLUTION(x, K)

1. $\mathbf{cx} \leftarrow \{\}$; $i \leftarrow 0$; $S_0 \leftarrow \textsc{base}(x, K)$
2. while $S_i \neq \emptyset$ do
3. sample $y_1, y_2, \ldots y_\beta$ from S_i
4. if sampling fails, output "fail" and exit.
5. $y \leftarrow \textsc{merge}(y_1, y_2, \ldots y_\beta)$
6. if $(\forall z)Q(x, y, z) = 1$ then return(y, "yes")
7. else find some z for which $Q(x, y, z) = 0$
8. $\mathbf{cx} \leftarrow \mathbf{cx} \cup \{z\}$
9. $S_{i+1} \leftarrow \{y \in S_i : Q(x, y, z) = 1\}$
10. $i \leftarrow i + 1$
11. end while
12. return("no")

As noted, by repetition we can reduce the failure probability of Step 3 to a negligible fraction. Accordingly, in the analysis that follows we will discount the probability that the algorithm fails in this step; formally we are just conditioning all probabilities on Step 3 succeeding every time it is executed.

Lemma 7.2 *If $T(n)$ is a bound on the number of repetitions of the while loop in FIND_SOLUTION(x, K), where $n = |x|$, then FIND_SOLUTION(x, K) runs in time* $\mathrm{poly}(T(n)) \cdot \mathrm{poly}(n)$ *with access to an Σ_{k-1}^{P} oracle.*

Proof: In Step 2, we need to check if the set S_i is empty. We note that

$$S_i = \{y \in \textsc{base}(x, K) : (\forall z \in \mathbf{cx})Q(x, y, z)\}.$$

By Property 2 of the base sets, $\textsc{base}(x, K)$ is recognizable in $\Sigma^{\mathrm{P}}_{k-2}$. In addition, the size of the set of counterexamples **cx** is bounded by $T(n)$ and Q is a $\Sigma^{\mathrm{P}}_{k-2}$ predicate, so we can determine if there exists some $y \in S_i$ with a single query to a $\Sigma^{\mathrm{P}}_{k-1}$ oracle.

By the above observations S_i is recognizable in $\mathrm{DTIME}(\mathrm{poly}(T(n)))^{\Sigma^{\mathrm{P}}_{k-2}}$. Therefore, in Step 3, by Theorem 7.1, we can sample uniformly from S_i with $\Sigma^{\mathrm{P}}_{k-1}$ oracle access. In Step 6, we can evaluate $(\forall z)Q(x, y, z)$ with one query to a $\Sigma^{\mathrm{P}}_{k-1}$ oracle because Q is a $\Sigma^{\mathrm{P}}_{k-2}$ predicate. Step 7 requires a simple binary search for some z such that $Q(x, y, z) = 0$; again, because Q is a $\Sigma^{\mathrm{P}}_{k-2}$ predicate, this is possible with access to a $\Sigma^{\mathrm{P}}_{k-1}$ oracle. ∎

Lemma 7.3 *Given an instance $x \in I_A$, FIND_SOLUTION(x, K) returns after at most $\alpha^{-1}\ln(|\textsc{base}(x, K)|)$ executions of the while loop, with probability at least* $1 - 1/(2\log B(x))$.

Proof: We claim that in the i-th repetition of the while loop, either (y, "yes) is returned in Step 6, or else $|S_{i+1}| \leq (1 - \alpha)|S_i|$ with very high probability. The size of set S_{i+1} depends only on the z chosen in Step 7. Call a z "bad" if $\Pr_{y \in S_i}[Q(x, y, z) = 0] \leq \alpha$; the bad z are precisely those that will result in S_{i+1} not shrinking sufficiently if they are chosen in Step 7.

Since y is the merge of β elements sampled uniformly from S_i, by Property 2 of the (α, β)-merger we have for all bad z:

$$\Pr[Q(x, y, z) = 0] \leq \frac{1}{2^{2p_z(|x|)+1}\log B(x)}.$$

There are at most $2^{p_z(|x|)}$ bad z when Step 7 is executed, so by the union bound,

$$\Pr[\exists \text{ bad } z \text{ in Step 7}] \leq \frac{1}{2^{p_z(|x|)+1}\log B(x)},$$

and we know that $\Pr[|S_{i-1}| > (1 - \alpha)|S_i|] = \Pr[\text{bad } z \text{ chosen}] \leq \Pr[\exists \text{ bad } z \text{ in Step 7}]$.

In a single call to FIND_SOLUTION, Step 7 is executed at most $2^{p_z(|x|)}$ times, since set **cx** grows by one with each execution, and when $\mathbf{cx} = \{0, 1\}^{p_z(|x|)}$, the procedure must return. Therefore, with probability at least $1 - \frac{1}{2\log B(x)}$, $|S_{i+1}| \leq (1 - \alpha)|S_i|$ for all i.

Since $S_0 = \textsc{base}(x, K)$, we can have at most $r = \alpha^{-1}\ln|\textsc{base}(x, K)|$ repetitions of the while loop, because after r repetitions,

$$|S_r| \leq (1 - \alpha)^r|S_0| < \left(\frac{1}{e}\right)^{\ln|\mathrm{BASE}(x,K)|} = 1,$$

which implies $S_r = \emptyset$, and the procedure must return. ∎

Lemma 7.4 *If FIND_SOLUTION returns (y, "yes"), then* $y \in \mathrm{sol}_A(x)$ *and* $m_A(x,y) \leq \beta K$*; if it returns "no", then* $\mathit{opt}_A(x) > K$.

Proof: If the procedure returns (y, "yes"), then $y \in \mathrm{sol}_A(x)$, because Step 6 verifies this fact. Then, by Property 1 of the (α, β)-merger, $m_A(x,y) \leq \beta K$.

If the procedure returns "no," then it must be that $S_i = \emptyset$. By Property 1 of the base sets, every $y \in \mathrm{sol}_A(x)$ with $m_A(x,y) \leq K$ appears in $\textsc{base}(x, K)$. Furthermore, by Equation 7.1, for every $y \in \mathrm{sol}_A(x)$, we have $(\forall z)Q(x,y,z)$. We conclude that if some $y \in \mathrm{sol}_A(x)$ has $m_A(x,y) \leq K$, then it must appear in set S_i for all i. Therefore $S_i = \emptyset$ implies $\mathrm{opt}_A(x) > K$. ∎

Theorem 7.5 *Define* $b(n)$ *and* $c(n)$ *as follows:*

$$b(n) = \max_{x:|x|=n} \log(B(x)) \quad c(n) = \max_{x:|x|=n, 1 \leq K \leq B(x)} (\ln |\textsc{base}(x, K)|).$$

Algorithm GENERIC_APPROX is a β*-approximation algorithm for* Σ_k^P *minimization problem A, that runs in time* $b(n) \cdot \mathrm{poly}(\alpha^{-1}c(n)) \cdot \mathrm{poly}(n)$ *with access to a* Σ_{k-1}^P *oracle, and succeeds with probability at least* $1/2$.

Proof: Given an input x of size n, the binary search executes at most $\log B(x) \leq b(n)$ calls to FIND_SOLUTION, and by Lemma 7.3, with probability at least $1/2$, every one of this calls returns after at most $T(n) = \alpha^{-1}c(n)$ repetitions of the while loop. By Lemma 7.2; the overall running time is $b(n) \cdot \mathrm{poly}(\alpha^{-1}c(n))$ with access to a $\Sigma_{k-1}^{\mathrm{P}}$ oracle.

Now, consider the binary search performed on K. If $K \geq \mathrm{opt}_A(x)$, then by Lemma 7.4 the procedure must return "yes"; therefore, the binary search cannot stop with $K > \mathrm{opt}_A(x)$. Therefore the binary search stops and outputs (y, "yes"),with $K \leq \mathrm{opt}_A(x)$, and by Lemma 7.4, this implies that $m_A(x,y) \leq \beta \mathrm{opt}_A(x)$. ∎

In the following sections, we use this framework to give approximation algorithms for a variety of minimization problems.

7.2 Approximation algorithms for k-mmww

In this section we use the generic framework to give near-optimal approximation algorithms for k-MMWW, for all $k \geq 2$. For simplicity, we will assume k is odd; a small modification will adapt the algorithm for even k.

Let C be a Π_{k-1} ND circuit that is an instance of k-MMWW. By replacing the $k-1$ groups of choice gates with variable sets $Y_1, Y_2, \ldots Y_{k-1}$, we obtain a deterministic circuit $C'(X, Y_1, Y_2, \ldots Y_{k-1})$ with the property that:

$$C \text{ accepts } X \Leftrightarrow (\forall Y_1)(\exists Y_2)\ldots(\forall Y_{k-1})C'(X, Y_1, Y_2, \ldots Y_{k-1}).$$

Let $n = |X|$ and $m = |Y_1|$. A solution for this instance is an n-bit string accepted by C, and the measure is the weight of the string; clearly n is an upper bound on this measure. We use a slightly non-obvious predicate for the required $\Sigma^{\mathrm{P}}_{k-2}$ predicate Q:

$$Q(C; u; v, v') \quad \Leftrightarrow \quad [(u \preceq v) \rightarrow (\exists Y_2)(\forall Y_3)\ldots(\forall Y_{k-1})C'(v, v', Y_2, Y_3, \ldots Y_{k-1})]$$

If there exists a solution u, then by monotonicity, $(\forall v, v')Q(C; u; v, v')$. Conversely, if $(\forall v, v')Q(C; u; v, v')$, then in particular, the right-hand-side must be true for all v' when $v = u$, which implies that C accepts u.

We define the base sets as follows: $B(C, K)$ is the set of all n-bit strings with weight at most K. This satisfies Property 1 and Property 2 for base sets. The (α, β)-merger MERGE is defined simply as the bit-wise OR of β n-bit strings, with

$$\alpha = \left(\frac{1}{2^{2(n+m)+1}\log n}\right)^{1/\beta}.$$

It is obvious that Property 1 for mergers holds. To verify Property 2 for mergers, we observe that for every v, v', $Q(C; u; v, v')$ implies $Q(C; u'; v, v')$ for all $u' \succeq u$. Therefore, $Q(C; \text{MERGE}(y_1, y_2, \ldots y_\beta); v, v') = 0$ if and only if $Q(C; y_i; v, v') = 0$ for all $i = 1 \ldots \beta$. We conclude that Property 2 holds for MERGE as we have defined it, for the α given above.

Theorem 7.6 *For every k, and for every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for k-MMWW that runs in time $2^{O(n+m)/\beta} \cdot \text{poly}(|C|)$ with a Σ^P_{k-1} oracle, and succeeds with probability at least $1/2$.*

Proof: Notice that $\ln|\text{BASE}(C, K)| \le \ln\binom{n}{K} \le O(K \ln n)$, and $K \le n$. Applying Theorem 7.5 with $b(n) \le \log n$ and $c(n) \le O(n \log n)$, we obtain the stated running time. ∎

7.3 Approximation algorithms for covering problems

We now adapt the generic algorithm for the five covering problems we have encountered: SUCCINCT SET COVER, IRREDUNDANT, SUCCINCT RICH HYPERGRAPH COVER, SUCCINCT TOURNAMENT DOMINATING SET, and SHORTEST IMPLICANT CORE. We begin with the algorithm for SUCCINCT SET COVER.

Let $\mathcal{C} = \{C_1, C_2, \ldots C_m\}$ be an instance, where each ϕ_i is a circuit over n variables. A solution for this instance is a subset $\mathcal{C}' \subseteq \mathcal{C}$ that covers all of $\{0,1\}^n$, and the measure is the size of the subset which can be no larger than m. For this problem, our predicate Q is the obvious one:

$$Q(\mathcal{C}, \mathcal{C}', x) \Leftrightarrow \bigvee_{C_i \in \mathcal{C}'} C_i(x).$$

The base sets are defined as follows: $B(\mathcal{C}, K)$ is the set of all subsets of $\mathcal{C}$ of size at most K – that is, every potential cover of size at most K appears in $B(\mathcal{C}, K)$. It is easy to see that Properties 1 and 2 for base sets hold.

Our (α, β)-merger MERGE returns the union of the β subsets it is given as input, and we define:

$$\alpha = \left(\frac{1}{2^{2n+1} \log m}\right)^{1/\beta}.$$

Property 1 for mergers is easy to verify. For Property 2, we notice that, as with the merger for k-MMWW, $Q(\mathcal{C}, \mathcal{C}', x)$ implies $Q(\mathcal{C}, \mathcal{C}'', x)$ for all $\mathcal{C}'' \supseteq \mathcal{C}'$. Therefore, $Q(\mathcal{C}, \text{MERGE}(C_1, C_2, \ldots C_\beta), x) = 0$ if and only if $Q(\mathcal{C}, C_i, x) = 0$ for all $i = 1 \ldots \beta$. Hence Property 2 holds for the α given above.

We note that all of the arguments in this section continue to hold if the C_i are 3-DNF formulae; therefore we have:

Theorem 7.7 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* SUCCINCT SET COVER *and* 3-DNF SUCCINCT SET COVER *that runs in time $2^{O(n)/\beta} \cdot \text{poly}(\sum_{i=1}^m |C_i|)$ with an NP oracle, and succeeds with probability at least $1/2$.*

Proof: Notice that $\ln |\text{BASE}(\mathcal{C}, K)| \leq \ln \binom{m}{K} \leq O(K \ln m)$, and $K \leq m$. Applying Theorem 7.5 with $b(m) \leq \log m$ and $c(m) \leq O(m \log m)$, we obtain the stated running time. ∎

It is relatively simple to adapt this algorithm for IRREDUNDANT. The arguments as to why the various required properties hold are essentially the same; the predicate Q, the base sets, and the merger are syntactically different, so we define them below. Recall that an instance of IRREDUNDANT is a DNF $\phi = t_1 \vee t_2 \vee \ldots \vee t_m$ over n variables, and a solution is a subset $I \subseteq [m]$ such that $(\vee_{i \in I} t_i) \equiv \phi$. The measure is the size of the subset I, which is clearly no larger than m. The predicate Q is defined as:

$$Q(\phi, I, x) \Leftrightarrow \left(\phi(x) \rightarrow \bigvee_{i \in I} t_i(x) \right)$$

A word of explanation is in order: because a formula consisting of a subset of the terms of ϕ cannot cover any assignment on which ϕ evaluates to zero, Q only needs to check that the subset of terms covers every assignment on which ϕ evaluates to one.

The base sets are defined as follows: $B(\phi, K)$ is the set of all subsets of $[m]$ of size at most K. The (α, β)-merger MERGE returns the union of the β subsets it is given as input, and we define:

$$\alpha = \left(\frac{1}{2^{2n+1} \log m} \right)^{1/\beta}.$$

These specifications yield the following approximation algorithm:

Theorem 7.8 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* IRREDUNDANT *that runs in time $2^{O(n)/\beta} \cdot \mathrm{poly}(|\phi|)$ with an NP oracle, and succeeds with probability at least $1/2$.*

Proof: The proof is essentially the same as the proof of Theorem 7.7. ∎

SUCCINCT RICH HYPERGRAPH COVER is essentially SUCCINCT SET COVER with a different encoding, so we can adapt the algorithm for SUCCINCT SET COVER quite easily. Recall that an instance of SUCCINCT RICH HYPERGRAPH COVER is a graph $G = (V, E)$ represented succinctly by a circuit $C_G(v, e)$ that outputs 1 if and only if hyperedge e includes vertex v. A solution is a subset $E' \subseteq E$ that covers the nodes, and the measure is the size of E', which can be no larger than $\log |V|$ because the hypergraph is rich. The predicate Q is defined as follows:

$$Q(C_G, E', v) \Leftrightarrow \bigvee_{e \in E'} C_G(v, e).$$

The base set $\mathrm{BASE}(C_G, K)$ is the set of all subsets of hyperedges with size at most K. The (α, β)-merger MERGE returns the union of the β subsets it is given as input, and we define:

$$\alpha = \left(\frac{1}{2^{2 \log |V| + 1} \log \log |V|} \right)^{1/\beta},$$

and we obtain the following approximation algorithm.

Theorem 7.9 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* SUCCINCT RICH HYPERGRAPH COVER *that runs in time* $2^{O(\log |V|)/\beta} \cdot \mathrm{poly}(|C_G|)$ *with an NP oracle, and succeeds with probability at least* $1/2$.

Proof: Notice that $\ln |\mathrm{BASE}(C_G, K)| \leq \ln \binom{|E|}{K} \leq O(K \ln |E|)$, and $K \leq \log |V|$. Applying Theorem 7.5 with $b(n) \leq \log\log |V|$ and $c(n) \leq O(\log |V| \log |E|)$, we obtain the stated running time. ■

Next, we describe the adaptation necessary for SUCCINCT TOURNAMENT DOMINATING SET, which again is straightforward. An instance of SUCCINCT TOURNAMENT DOMINATING SET is a tournament graph $G = (V, E)$ represented succinctly by a circuit $C_G(x, y)$ which outputs 1 if and only if $(x, y) \in E$. A solution is a dominating set $V' \subseteq V$; the measure is the size of V', and we know that this is never larger than $\log |V|$. Our predicate Q is defined below:

$$Q(C_G, V', v) \Leftrightarrow \bigvee_{v' \in V'} (v = v') \vee C_G(v', v).$$

The base set $\mathrm{BASE}(C_G, K)$ is the set of all subsets of vertices with size at most K. The (α, β)-merger MERGE returns the union of the β subsets it is given as input, and we define:

$$\alpha = \left(\frac{1}{2^{2\log |V|+1} \log\log |V|}\right)^{1/\beta},$$

and we obtain the following approximation algorithm.

Theorem 7.10 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* SUCCINCT TOURNAMENT DOMINATING SET *that runs in time* $2^{O(\log |V|)/\beta} \cdot \mathrm{poly}(|C_G|)$ *with an NP oracle, and succeeds with probability at least* $1/2$.

Proof: Notice that $\ln |\mathrm{BASE}(C_G, K)| \leq \ln \binom{|V|}{K} \leq O(K \ln |V|)$, and $K \leq \log |V|$. Applying Theorem 7.5 with $b(n) \leq \log\log |V|$ and $c(n) \leq O(\log^2 |V|)$, we obtain the stated running time. ■

Finally, we present an algorithm for SHORTEST IMPLICANT CORE. Recall that an instance of SHORTEST IMPLICANT CORE is a DNF ϕ over n variables and an implicant C of ϕ. A solution is an implicant $C' \subseteq C$, and the measure is the size of C', which is clearly always n or smaller. Our predicate Q is the obvious one:

$$Q(\phi, C; C'; x) \Leftrightarrow (C'(x) \to \phi(x)).$$

The base set $\textsc{base}(\phi, C; K)$ includes all subsets $C' \subseteq C$ of size at most K. The (α, β) merger MERGE takes the conjunction of the β conjunctions it is given as input. Because these are all subsets of the literals of C, they are pairwise consistent, so we never end up with a conjunction containing both a variable and its negation. Noticing that for any fixed x, $Q(\phi, C; C'; x)$ implies $Q(\phi, C; C''; x)$ for all $C' \subseteq C'' \subseteq C$, we see that this merger fulfills the required properties with

$$\alpha = \left(\frac{1}{2^{2n+1}\log n}\right)^{1/\beta}.$$

Theorem 7.11 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* SHORTEST IMPLICANT CORE *that runs in time* $2^{O(n)/\beta} \cdot \mathrm{poly}(|\phi|)$ *with an NP oracle, and succeeds with probability at least* $1/2$.

Proof: Notice that $\ln|\textsc{base}(\phi, C; K)| \leq \ln\binom{n}{K} \leq O(K \ln n)$, and $K \leq n$. Applying Theorem 7.5 with $b(n) \leq \log n$ and $c(n) \leq O(n \log n)$, we obtain the stated running time. ■

7.4 Approximation algorithms for formula and circuit minimization

The generic algorithm can also be adapted to give approximation algorithms for the formula and circuit variants of MIN DNF. Both of these problems are in Σ_2^p and suspected to be Σ_2^p-complete; however proving completeness remains an open problem. Nevertheless, we press ahead with the approximation algorithms. The problem definitions follow:

MINIMUM EQUIVALENT FORMULA (MIN FORMULA):

INSTANCE: A DeMorgan formula ϕ.

SOLUTION: A DeMorgan formula ϕ' for which $\phi' \equiv \phi$.

MEASURE: The number of leaves of formula ϕ'.

MINIMUM EQUIVALENT CIRCUIT (MIN CIRCUIT):

INSTANCE: A circuit ϕ.

SOLUTION: A circuit ϕ' for which $\phi' \equiv \phi$.

MEASURE: The size of the circuit ϕ'.

Here we are essentially adapting learning algorithms already known in [14] for our purposes. In the next two lemmas, we adapt a lemma from [14] concerning two *amplifier* functions that appear originally in [11] and [40]:

Lemma 7.12 *For any probability p, $\beta > 0$ and $0 \leq \alpha \leq \frac{1}{2e}p^{2/\beta}$, the following holds for $b \in \{0,1\}$: if Boolean variables $y_1, y_2, \ldots y_\beta$ are independently set to b with probability α, then $\Pr[maj(y_1, y_2, \ldots y_\beta) = b] \leq p$.*

Proof: Let $X_1, X_2, \ldots X_\beta$ be indicator random variables for the events $(y_1 = b), \ldots (y_\beta = b)$, and let $X = \sum X_i$. Let $\mu = E[X]$. By Chernoff bounds (this specific version is proved in [80]), we have:

$$\Pr[X \geq \gamma\mu] \leq \exp(-(\ln\gamma + \frac{1}{\gamma} - 1)\gamma\mu).$$

In our case, we have $\mu = \alpha\beta$.

$$\begin{aligned}
\Pr[\mathrm{maj}(y_1, y_2, \ldots y_\beta) = b] &= \Pr[X \geq \beta/2] \\
&\leq \Pr[X \geq \frac{1}{2\alpha}\mu] \\
&\leq \exp(-(2\alpha + \ln(\frac{1}{2\alpha}) - 1)(\beta/2)) \\
&\leq \left(\frac{2e\alpha}{e^{2\alpha}}\right)^{\beta/2} \leq (2e\alpha)^{\beta/2} \leq p.
\end{aligned}$$

■

Lemma 7.13 *Define the function*

$$A(y_1, y_2, \ldots y_\beta) = \bigwedge_{i=1}^{\sqrt{\beta}} \bigvee_{j=1}^{\sqrt{\beta}} y_{(i \cdot j)}.$$

For any probability p, $\beta > 0$ and $0 \leq \alpha \leq (p/\sqrt{\beta})^{1/\sqrt{\beta}}$, the following holds for $b \in \{0,1\}$: if Boolean variables $y_1, y_2, \ldots y_\beta$ are independently set to b with probability at most α, then $\Pr[A(y_1, y_2, \ldots y_\beta) = b] \leq p$.

Proof: Consider the case $b = 1$. The probability a fixed OR gate will compute one is at most $\sqrt{\beta}\alpha$ by the union bound, and the probability that all of the OR gates compute one (and hence the formula outputs one) is at most $(\sqrt{\beta}\alpha)^{\sqrt{\beta}} \leq p$, as required.

If $b = 0$, then the probability that a fixed OR gate outputs zero is at most $\alpha^{\sqrt{\beta}}$; the probability that all of the OR gates output zero (and hence the formula outputs zero) is at most $\sqrt{\beta}\alpha^{\sqrt{\beta}} \leq p$, as required. ∎

We proceed with the algorithm for MIN FORMULA. An instance of this problem is a DeMorgan formula ϕ over n variables, and a solution is a DeMorgan formula ϕ' equivalent to ϕ. The measure is the size of formula ϕ', which is clearly bounded by $|\phi|$. The predicate Q is the obvious one:

$$Q(\phi, \phi', x) \Leftrightarrow \phi'(x) = \phi(x).$$

The base sets $\text{BASE}(\phi, K)$ are defined as follows: $\text{BASE}(\phi, K)$ is the set of all DeMorgan formulae over n variables of size at most K. The (α, β)-merger MERGE takes its β input formulas $\phi_1, \ldots \phi_\beta$, and returns the formula $A(\phi_1, \ldots \phi_\beta)$, where A is the formula in Lemma 7.13. Property 1 for mergers clearly holds; to verify property 2, we use Lemma 7.13. We need to choose

$$\alpha = \left(\frac{1}{2^{2n+1} \log |\phi| \sqrt{\beta}} \right)^{1/\sqrt{\beta}}.$$

Suppose for some fixed $S \subseteq \text{BASE}(\phi, K)$ and $x \in \{0,1\}^n$, we have $\Pr_{\phi' \in S}[Q(\phi, \phi', x) = 0] \leq \alpha$. Letting $b = \phi(x)$, we see that this implies that $\Pr_{\phi' \in S}[\phi'(x) = b] \leq \alpha$. Then by Lemma 7.13, we have

$$\Pr_{\phi_1, \ldots \phi_\beta \in S}[A(\phi_1(x), \ldots \phi_\beta(x)) = b] \leq \frac{1}{2^{2n+1} \log |\phi|},$$

as required. Therefore, Property 2 for mergers holds, for the choice of α above. We have:

Theorem 7.14 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* MIN FORMULA *that runs in time $2^{O(n)/\sqrt{\beta}} \cdot \text{poly}(|\phi|)$ with an NP oracle, and succeeds with probability at least $1/2$.*

Proof: Notice that $\ln |\text{BASE}(\phi, K)| \leq O(K)$, and $K \leq |\phi|$. Applying Theorem 7.5 with $b(n) \leq |\phi|$ and $c(n) \leq O(|\phi|)$, we obtain the stated running time. ∎

For MIN CIRCUIT, the algorithm is quite similar; the only difference is that we replace the merger above with a function MERGE that returns a circuit computing the majority of the β circuits it receives as inputs. The majority function has $O(nlogn)$-size circuits, and therefore the MERGE of β circuits, each of size at most K yields a circuit of size at most $O(\beta \log \beta) + \beta K$. Setting

$$\alpha = \left(\frac{1}{e 2^{2n+2} \log |C|} \right)^{2/\beta},$$

and applying Lemma 7.12 shows that a slight relaxation of property 2 holds. This is still strong enough to obtain the following theorem:

Theorem 7.15 *For every $\beta \leq 2^n$, algorithm GENERIC_APPROX, specialized as described above, is a $O(\beta)$-approximation algorithm for* MIN CIRCUIT *that runs in time $2^{O(n)/\beta} \cdot \mathrm{poly}(|C|)$ with an NP oracle, and succeeds with probability at least* $1/2$.

Proof: The proof is essentially the same as the proof of Theorem 7.14 above; however we need to observe that without loss of generality, circuit C depends on all n variables, and therefore has size at least n. This implies that the merge of β circuits of size at most K produces a circuit of size at most $O(\beta)K$ (because $\log \beta \leq n$). A slight modification of Lemma 7.4 and Theorem 7.5 gives the desired result. ■

7.5 An approximation algorithm for DNF minimization

For the two DNF minimization problems MIN DNF and MIN TERM DNF we do not know how to adapt the generic algorithm to obtain an efficient approximation algorithm. The difficulty lies in defining polynomial-time recognizable base sets containing all DNFs of a certain size *that imply the target DNF*, as we did for IRREDUNDANT.

What we can achieve is an approximation algorithm like the ones above whose running time depends on the size of the input DNF ϕ *and the size of the size of the smallest CNF that computes* ϕ. For certain function (like those with small decision trees), this may be a sufficiently efficient algorithm.

The approximation algorithm for MIN DNF has two phases. In the first phase, we construct a CNF representation of the input DNF ϕ; in the second phase, we use the CNF representation to find a short equivalent DNF. We make use of the following definition and lemma:

Definition 7.1 *Let ϕ be a CNF (resp. DNF) formula. A* transversal *of ϕ is a conjunction (resp. disjunction) C with one literal chosen from each clause (resp. term) of ϕ.*

Lemma 7.16 *Let ϕ be a CNF (resp. DNF) formula. Then every prime implicant (resp. implicate) of ϕ is a transversal of ϕ. Similarly, any transversal of ϕ is an implicant (resp. implicate) of ϕ.*

Proof: Suppose some prime implicant C is not a transversal of CNF ϕ. Then either C excludes all of the literals from some clause, or else C is some transversal with additional literals added. In the latter case C is not prime; in the former, we can extend C into an assignment A for which $C(A) = 1$, but A fails to satisfy some clause of ϕ, and hence $\phi(A) = 0$, a contradiction.

The other direction is clear, and the argument for DNFs and implicates is identical.

■

Notice that given a CNF or DNF, it is easy to recognize a transversal. We use this idea as follows: we are given a DNF ϕ that we wish to minimize. We first find the (approximately) smallest CNF that computes ϕ, using the generic algorithm once again. Having the CNF in hand, we can easily recognize candidate DNFs of size at most K that are implied by ϕ, because we can recognize transversals. These form our base sets, and the analysis is then similar to that of irredundant.

We begin by describing the first phase of the algorithm, which produces an equivalent CNF from the input DNF ϕ. A feasible solution to the optimization problem we are solving in the first phase is a CNF ϕ' that is equivalent to ϕ; the measure is the size of ϕ', which is bounded by $n2^n$. We define the predicate Q below:

$$Q(\phi, \phi', x) \Leftrightarrow \phi(x) = \phi'(x).$$

The base sets are defined as follow. If we are solving MIN TERM DNF, BASE(ϕ, K) consists of all conjunctions of K or fewer transversals of ϕ; if we are solving MIN DNF, BASE(ϕ, K) consists of all CNFs with at most K literals, and with every clause being a transversal of ϕ. Clearly this satisfies properties 1 and 2 for base sets. We need to use a slight modification of the generic algorithm: instead of performing a binary search on K, we search linearly, starting with $1, 2, \ldots$, and ending with $K \leq L$, where L is the size of the smallest CNF equivalent to ϕ. We need a slightly modified version of Lemma 7.3 to hold; namely the success probability should be $1 - \frac{1}{2L}$; this is accomplished by simply making α larger. Our (α, β)-merger MERGE takes the AND of its β input CNFs, with

$$\alpha = \left(\frac{1}{2^{2n+1}L}\right)^{1/\beta}.$$

As with IRREDUNDANT, we note that $Q(\phi, \text{MERGE}(\phi_1, \ldots \phi_\beta), x) = 0$ implies $Q(\phi, \phi_i, x) = 0$ for all $i = 1 \ldots \beta$. This is true because the only way $Q(\phi, \phi', x) = 0$ is if $\phi(x) = 0$ and $\phi'(x) = 1$, since every $\phi' \in \text{BASE}(\phi, K)$ is implies by ϕ.

Notice that $\ln |\text{BASE}(\phi, K)| \leq nK \ln m$, if ϕ has m terms over n variables. Applying a slightly modified Theorem 7.5 (to account for the linear search instead of binary search), we conclude that the first phase of the algorithm runs in time $2^{O(n)/\beta} \cdot \text{poly}(L, n, \log m)$, and returns, with probability at least $1/2$, a CNF ϕ' equivalent to ϕ of size at most βL.

In phase 2, we essentially repeat phase 1, using ϕ' as an input, and producing a small DNF ϕ'' equivalent to ϕ. Formally, we can use the algorithm just described, without modification, to obtain $\neg\phi''$ from input $\neg\phi'$. Finally, we have the following theorem:

Theorem 7.17 *For every β, algorithm GENERIC_APPROX, specialized as described above, is a β-approximation algorithm for* MIN DNF *and* MIN TERM DNF *that runs in time $2^{O(n)/\beta} \cdot \text{poly}(|\phi|, L)$ with an NP oracle, where L is the size of the smallest equivalent CNF, and succeeds with probability at least* $1/2$.

7.6 Summary of approximation schemes

As is apparent from the theorems in the previous section, our approximation algorithms are actually approximation *schemes*: they can achieve any approximation ratio β, but of course there is a tradeoff with the running time. In Table 7.1 we summarize the approximation ratios achievable in (expected) polynomial time.

It is reasonable to ask whether a better ratio of perhaps $N/\log^2 N$ is possible without degrading the running time by more than a polynomial factor (our algorithms already achieve this approximation quality in *quasipolynomial* time). More ambitiously, can we achieve a ratio of $N^{1-f(N)}$ for any $f(N) = o(1)$? Using the inapproximability results from previous chapters, under a strong complexity assumption, we can show limits on the lower order terms of the approximation ratio achievable.

problem	size measure N	ratio
k-MMWW	circuit size	$\epsilon \frac{N}{\log N}$
SUCCINCT RICH HYPERGRAPH COVER	circuit size	$\epsilon \frac{N}{\log N}$
SUCCINCT TOURNAMENT DOMINATING SET	circuit size	$\epsilon \frac{N}{\log N}$
SUCCINCT SET COVER	# variables	$\epsilon \frac{N}{\log N}$
3-DNF SUCCINCT SET COVER	# variables	$\epsilon \frac{N}{\log N}$
IRREDUNDANT	# variables	$\epsilon \frac{N}{\log N}$
MIN TERM DNF (*)	# variables	$\epsilon \frac{N}{\log N}$
MIN DNF (*)	# variables	$\epsilon \frac{N}{\log N}$
MIN FORMULA	# variables	$\epsilon \left(\frac{N}{\log N}\right)^2$
MIN CIRCUIT	# variables	$\epsilon \frac{N}{\log N}$

Table 7.1: Approximation algorithms running in expected polynomial time with access to an NP oracle. ϵ is any constant greater than zero. The entries marked with (*) run in expected polynomial time in the size of the input DNF *and* the size of the shortest equivalent CNF.

Theorem 7.18 *If ZPTIME(subexp)^{NP} does not contain Σ_2^p, then there is no $N^{1-f(N)}$-approximation algorithm for* SHORTEST IMPLICANT CORE, SUCCINCT SET COVER, 3-DNF SUCCINCT SET COVER, IRREDUNDANT, *or* SUCCINCT RICH HYPERGRAPH COVER *that runs in expected $T(N) = 2^{N^{o(f(N)^2)}}$ time with access to an NP oracle.*

Proof: We prove the result for SUCCINCT SET COVER; the proof is similar for the other problems, using the appropriate reduction from Chapter 5. By Theorems 4.5 and 5.3, there exists a reduction from QSAT$_2$ to SUCCINCT SET COVER WITH GAP $g(N)$ that maps instances of QSAT$_2$ with size n to instances of SUCCINCT SET COVER with size $N = \text{poly}(n^{f(N)^{-2}})$, and gap $g(N) = N^{1-f(N)}$. Now, suppose an algorithm for SUCCINCT SET COVER achieved a ratio of $g(N)$. If it runs in expected time $T(N)$, then it runs in expected subexponential time in n, because

$$T(N) = \exp(n^{O(f(N)^{-2})\cdot o(f(N)^2)}) = \exp(n^{o(1)}) = \text{subexp}(n).$$

■

For a concrete example: although we can achieve a ratio of $N/\log N$ for these problems in polynomial time, the above theorem says that this cannot be improved to $N/(\log N^{\log^{1/2} N})$ (take $f(N) = \log\log N/\log^{1/2} N$ in the above theorem, and notice that $\text{poly}(N) = 2^{N^{o(f(N)^2)}}$).

Improving the ratio to $N/\log^2 N$, or proving that such a ratio is impossible under a complexity assumption remains an interesting open problem.

Chapter 8

Conclusions and Open Problems

In this dissertation, we have attempted to provide a solid theoretical foundation for understanding practically important problems in logic minimization, and DNF minimization in particular. Toward this end we have established Σ_2^p-completeness for DNF minimization and several related optimization problems, established very large inapproximability ratios for these problems, and described the first approximation algorithms for logic minimization with guaranteed approximation ratios.

Perhaps more significantly, we have established general tools and results that should facilitate similar investigations of optimization problems in the PH. Specifically, we have presented an assortment of natural optimization problems and basic combinatorial problems that are complete for various levels of the PH, which should provide a starting point for future reductions.

We have developed techniques for proving strong inapproximability results for problems in the PH that relies on explicit combinatorial constructions instead of Probabilistically Checkable Proofs, and, using these techniques, we have given inapproximable minimization and maximization problems for every level of the hierarchy, which should provide a starting point for future proofs of inapproximability.

The results concerning SHORTEST IMPLICANT[DNF] from Chapter 6 are theoretically interesting, as they give the first example of a natural problem above NP requiring only limited nondeterminism. This classification also provides important practical insights: for example, it is (probably) hopeless to design a approximation algorithm for DNF minimization achieving a good approximation ratio using a subroutine that finds the shortest implicant of DNF formulae.

The approximation algorithm framework in Chapter 7 is quite general, and we have adapted it to give approximation schemes for many of the minimization problems in the PH that we have encountered. These approximation schemes trade off running time for approximation quality, and are, to our knowledge, the first approximation algorithms with formal approximation guarantees for these problems. The generic framework should be useful in the future for developing approximation algorithms for other minimization problems in the PH.

We end with a sampling of the open problems, grouped by chapter.

Chapter 2. Prove that MIN CIRCUIT or MIN FORMULA are Σ_2^p-complete. As a first step, prove that the depth-3 formula minimization problem is Σ_2^p-complete.

Chapter 3. Find additional applications for or-codes, including possibly strengthening or even replacing certain PCP-based inapproximability results at the NP level. Build better dispersers. Improvements relevant to the applications in this thesis have been obtained very recently [79], but for certain choices of parameters, optimal explicit constructions are still not known.

Chapter 4. Sharpen the approximation hardness results in a number of directions: improve the result for k-MMWW when the size measure is circuit size to get the reduction to run in poly$(sn^{\epsilon^{-1}})$ time (instead of poly$(sn^{\epsilon^{-2}})$). Devise a self-improvement transformation for k-MMZ, to obtain strong $n^{1-\epsilon}$ ratios in terms of circuit size for that problem. Modify the gap-producing reduction for 1-MMWW so that the circuit produced is a monotone circuit (this would translate to improved inapproximability ratios for minimum propositional proof length problems via the reductions in [2]); if the circuit produced could be made to be depth 3, this would be a major result as it would translate into improved inapproximability ratios for label-cover, and a host of other NP optimization problems.

Chapter 5. Tighten the reductions for the problems and size measures listed in the bottom four sections of Table 5.1, to achieve $N^{1-\epsilon}$ ratios. Settle the complexity of the two maximization problems in Section 5.10.

Chapter 6. Find non-trivial approximation algorithms for the formula and circuit versions of the shortest implicant problem. Clarify the relationship of the class

$\mathrm{GC}(\log^2 n, \mathrm{coNP})$ to surrounding classes; as a starting point, prove that it does not contain NP, under a plausible complexity assumption. Identify other natural complete problems for $\mathrm{GC}(\log^2 n, \mathrm{coNP})$ or its complement.

Chapter 7. Improve the approximation algorithms so that they achieve an $n/\log^2$ ratio (or better), or show that such an improvement is not possible under a plausible complexity assumption. Derandomize the generic approximation algorithm or give a P^{NP} approximation algorithm for any of the Σ_2^p-complete problems considered. Find a non-trivial approximation algorithm for MIN DNF that runs in polynomial time. Finally, it would be interesting to investigate the practicality of implementing the approximation algorithms (particularly for MIN FORMULA, where heuristics do not perform nearly as well as for MIN DNF), and the empirical performance of such implementations.

In general, many of the problems treated in this dissertation represent a particularly striking example of the disparity between worst-case analysis and real-world performance. Despite the spectacular hardness of these problems from a complexity perspective, the logic synthesis community routinely deals with real-world instances quite successfully. An understanding of this phenomenon (what special cases are easier; why certain heuristics perform well on average) is perhaps the most important outstanding challenge here, and in many other settings.

Bibliography

[1] M. Agrawal and T. Thierauf. The Boolean isomorphism problem. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS '96)*, pages 422–30, 1996.

[2] M. Alekhnovich, S. Buss, S. Moran, and T. Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. In *Mathematical Foundations of Computer Science, 23rd International Symposium (MFCS 98)*, pages 176–184, Brno, Czech Republic, August 1998. Springer-Verlag. To appear in Journal of Symbolic Logic.

[3] S. Arora. Approximability of NP-hard problems. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC 98)*, Dallas, Texas, May 1998.

[4] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of $\mathcal{NP}$-hard problems. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 284–293, Las Vegas, Nevada, 29 May–1 June 1995.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[6] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, Jan. 1998.

[7] S. Arora and M. Sudan. Improved low degree testing and its applications. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 97)*, pages 485–495, El Paso, Texas, 4–6 May 1997.

[8] M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using

an NP-oracle. Technical Report TR98-032, Electronic Colloquium on Computational Complexity, 1998.

[9] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC 93)*, pages 294–304, San Diego, California, 16–18 May 1993.

[10] E. Ben-Sasson. personal communication.

[11] R. B. Boppana. Amplification of probabilistic Boolean formulas. *Advances in Computing Research*, 5(4):27–45, 1989.

[12] B. Borchert and D. Ranjan. The circuit subfunction relations are Σ_2^p-complete. Technical Report MPI-I-93-121, MPI Saarbrücken, 1993.

[13] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on Boolean functions. *Theory of Computing Systems*, 31(6):679–693, 1998.

[14] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, June 1996.

[15] S. Bublitz, U. Schürfeld, I. Wegener, and B. Voigt. Properties of complexity measures for PRAMs and WRAMs. *Theoretical Computer Science*, 48(1):53–73, 1986.

[16] L. Cai and J. Chen. On the amount of nondeterminism and the power of verifying. *SIAM Journal on Computing*, 26(3):733–750, 1997.

[17] A. Condon, J. Feigenbaum, C. Lund, and P. Shor. Probabilistically checkable debate systems and approximation algorithms for PSPACE-hard functions (extended abstract). In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC 93)*, pages 305–314, San Diego, California, 16–18 May 1993.

[18] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 97)*, pages 151–158, 1971.

[19] O. Coudert. Two-level logic minimization: an overview. *Integration, the VLSI journal*, 17(2):97–140, 1994.

[20] O. Coudert and J. C. Madre. Metaprime: an interactive fault-tree analyzer. *IEEE Transactions on Reliability*, 43(1):121–127, March 1994.

[21] P. Crescenzi, R. Silvestri, and L. Trevisan. To weight or not to weight: Where is the question? In *Proceedings of the 4th Israeli Symposium on Theory of Computing and Systems*, pages 68–77. IEEE, 1996.

[22] S. L. A. Czort. The complexity of minimizing disjunctive normal form formulas. Master's thesis, University of Aarhus, Denmark, 1999.

[23] S. Devadas, A. Ghosh, and K. Keutzer. *Logic Synthesis*. McGraw-Hill, Inc., 1994.

[24] J. Diaz and J. Toran. Classes of bounded nondeterminism. *Mathematical Systems Theory*, 23(1):21–32, 1990.

[25] I. Dinur and S. Safra. On the hardness of approximating label cover. Technical Report TR99-015, Electronic Colloquium on Computational Complexity, 1999.

[26] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, Nov. 1996.

[27] J. F. Gimpel. A method of producing a boolean function having an arbitrarily prescribed prime implicant table. *IEEE Transactions on Computers*, 14:485–488, 1965.

[28] O. Goldreich, A. Sahai, and S. Vadhan. Can statistical zero-knowledge be made non-interactive?, or On the relationship of SZK and NISZK. Technical Report TR99-013, Electronic Colloquium on Computational Complexity, May 1999.

[29] O. Goldreich and S. Vadhan. Comparing entropies in statistical zero-knowledge with applications to the structure of SZK. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, Atlanta, GA, May 1999. IEEE.

[30] M. H. Goldwasser and R. Motwani. Intractability of assembly sequencing: Unit disks in the plane. In *Proceedings of the Workshop on Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*, pages 307–320. Springer-Verlag, 1997.

[31] V. Guruswami and M. Sudan. List decoding algorithms for certain concatenated codes. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC 00)*, 2000.

[32] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *37th Annual Symposium on Foundations of Computer Science*, pages 627–636, Burlington, Vermont, 14–16 Oct. 1996. IEEE.

[33] E. Hemaspaandra and G. Wechsung. The minimization problem for Boolean formulas. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 575–84, 1997.

[34] L. A. Hemaspaandra. Raising np lower bounds to parallel lower bounds. *SIGACT News*, 28(2):2–13, June 1997.

[35] D. Hochbaum, editor. *Approximation Algorithms for NP-hard problems.* PWS Publishing Co., 1997.

[36] D. T. Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^p-complete. *Theoretical Computer Science*, 33(2–3):305–326, Oct. 1984.

[37] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2–3):169–188, 1986.

[38] D. S. Johnson. Approximation algorithms for combinatorial problems. In *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 38–49, Austin, Texas, 30 Apr.–2 May 1973.

[39] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, May 2000.

[40] S. K. Kannan. On the query complexity of learning. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 58–66, Santa Cruz, California, 26–28 July 1993. ACM, ACM Press.

[41] R. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[42] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing (STOC 80)*, pages 302–309, 1980.

[43] C. M. R. Kintaal and P. C. Fischer. Refining nondeterminism in relativized polynomial-time bounded c omputations. *SIAM Journal on Computing*, 9(1):46–53, 1980.

[44] M. Kiwi, C. Lund, A. Russell, D. Spielman, and R. Sundaram. Alternation in interaction. In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 294–303, Amsterdam, The Netherlands, 28 June–1 July 1994. IEEE, IEEE Computer Society Press.

[45] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, Atlanta, GA, May 1999. IEEE.

[46] K.-I. Ko and C.-L. Lin. Non-approximability in the polynomial-time hierarchy. Technical Report TR-94-2, Department of Computer Science, State University of New York at Stony Brook, 1994.

[47] K.-I. Ko and C.-L. Lin. On the complexity of min-max optimization problems and their approximation. In D.-Z. Du and P. M. Pardalos, editors, *Minimax and Applications*, pages 219–239. Kluwer Academic Publishers, 1995.

[48] K.-I. Ko and C.-L. Lin. On the longest circuit in an alterable digraph. *Journal of Global Optimization*, 7(3):279–295, 1995.

[49] K.-I. Ko and W.-G. Tzeng. Three Σ_2^{p}-complete problems in computational learning theory. *Computational Complexity*, 1(3):269–310, 1991.

[50] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.

[51] P. G. Kolaitis and M. N. Thakur. Approximation properties of NP minimization classes. *Journal of Computer and System Sciences*, 50(3):391–411, June 1995.

[52] L. Levin. Universal sorting problems. *Problemy Peredaci Informacii*, 9:265–266, 1973.

[53] C.-L. Lin. *Intractibility in the Approximation of Combinatorial Optimization Problems.* PhD thesis, State University of New York at Stony Brook, 1995.

[54] C.-L. Lin. Optimizing TRIEs for ordered pattern matching is Π_2^p-complete. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (STC 95)*, 1995.

[55] W. J. Masek. Some NP-complete set covering problems. unpublished manuscript, 1979.

[56] E. J. McCluskey. Minimization of Boolean functions. *Bell Systems Technical Journal*, 35(5):1417–44, 1956.

[57] N. Megiddo and U. Vishkin. On finding a minimum dominating set in a tournament. *Theoretical Comput. Sci.*, 61(2–3):307–316, Nov. 1988.

[58] A. Meyer and L. J. Stockmeyer. The equivlence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129, 1972.

[59] A. Narayan. Recent advances in BDD based representations for Boolean functions: a survey. In *Proceedings Twelfth International Conference on VLSI Design*, pages 408–413, 1999.

[60] N. Nisam and A. Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1):148–173, Feb. 1999. Preliminary versions in [78] and [61].

[61] N. Nisan. Extracting randomness: How and why: A survey. In *Proceedings, Eleventh Annual IEEE Conference on Computational Complexity*, pages 44–58, Philadelphia, Pennsylvania, 24–27 May 1996. IEEE, IEEE Computer Society Press.

[62] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[63] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, Dec. 1991.

[64] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V–C dimension. *Journal of Computer and System Sciences*, 53(2):161–70, 1996.

[65] C. H. Papadimitriou and S. K. Zachos. Two remarks on the power of counting. In *Proceedings of Theoretical Computer Science. 6th GI Conference*, pages 269–275, Dortmund, West Germany, 5–7Jan. 1983.

[66] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, Oct. 1988.

[67] W. V. Quine. The problem of simplifying truth functions. *American Mathematics Monthly*, 59(8):521–531, 1952.

[68] J. Radhakrishnan and A. Ta-Shma. Tight bounds for depth-two superconcentrators. In *38th Annual Symposium on Foundations of Computer Science (FOCS 97)*, pages 585–594, Miami Beach, Florida, 20–22 Oct. 1997.

[69] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st ACM Symposium on Theory of Computing (STOC 99)*, 1999.

[70] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 97)*, pages 475–484, El Paso, Texas, 4–6 May 1997.

[71] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

[72] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with union and difference operators. *J. ACM*, 27(4):633–655, October 1980.

[73] M. Schaefer. Deciding the Vapnik-Cervonenkis dimension is Σ_3^p-complete. *Journal of Computer and System Sciences*, 58(1):177–182, Feb. 1999.

[74] M. Schaefer. Graph Ramsey theory and the polynomial hierarchy. In *Proceedings of the 14th annual conference on Computational Complexity*. IEEE Computer Society Press, 1999.

[75] M. Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, June 1988.

[76] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28(4):1433–1459, 1999.

[77] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, Oct. 1976.

[78] A. Ta-Shma. Almost optimal dispersers. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC 98)*, pages 196–202, Dallas, Texas, May 1998.

[79] A. Ta-Shma, C. Umans, and D. Zuckerman. Unbalanced expanders and improved extractors and dispersers. In preparation., 2000.

[80] L. Trevisan. Construction of extractors using pseudorandom generators. In *Proceedings of the 31st ACM Symposium on Theory of Computing (STOC 99)*, 1999.

[81] C. Umans. The minimum equivalent DNF problem and shortest implicants. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS 98)*, pages 556–563, 1998.

[82] C. Umans. Hardness of approximating Σ_2^p minimization problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 99)*, 1999.

[83] C. Umans. On the complexity and inapproximability of shortest implicant problems. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium (ICALP 99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 687–696, Prague, Czech Republic, July 1999. Springer-Verlag.

[84] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer-Verlag, 1999.

[85] K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1–2):53–80, 1987.

[86] K. W. Wagner. The complexity of combinatorial problems with succinct input representations. *Acta Informatica*, 23(3):325–356, 1986.

[87] I. Wegener. *The complexity of Boolean functions.* Teubner, 1987.

[88] H. Zantema and H. Bodlaender. Finding small equivalent decision trees is hard. Technical Report UU-CS-1999-02, Department of Computer Science, Utrecht University, 1999.

[89] D. Zuckerman. On unapproximable versions of *NP*-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, Dec. 1996.