The Complexity of Optimization Problems

Mark William Krentel
Ph. D. Thesis

87-834
May 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

THE COMPLEXITY OF OPTIMIZATION PROBLEMS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Mark William Krentel

May 1987

THE COMPLEXITY OF OPTIMIZATION PROBLEMS

Mark William Krentel, Ph.D.

Cornell University 1987

We study computational complexity theory and define a class of optimization problems called OptP (Optimization Polynomial Time), and we show that TRAVELLING SALESPERSON, KNAPSACK and 0-1 INTEGER LINEAR PROGRAMMING are complete for OptP. OptP is a natural generalization of NP (Nondeterministic Polynomial Time), but while NP only considers problems at the level of their yes/no question, the value of an OptP function is the optimal value of the problem. This approach enables us to show a deeper level of structure in these problems than is possible in NP.

OptP is a subset of FPSAT, the class of functions computable in polynomial time with an oracle for NP. Our central result is that any FPSAT function decomposes into an OptP function followed by polynomial-time computation. The significance of this result is that it quantifies "how much" NP-completeness is in a problem, i.e., the number of NP queries

it takes to compute the function. It also allows us to unify the classes NP, DP and DELTA-2 in a natural way. For example, we prove that an OptP-completeness result implies, as corollaries, NP-, DP- and DELTA-2-completeness results.

We also prove separation results on subclasses of FPSAT by restricting the number of calls to the NP oracle. For example, TRAVELLING SALESPERSON is complete for O(n) queries, CLIQUE is complete for O(log n) queries and BIN PACKING can be solved in O(log log n) queries. We prove these classes distinct under the assumption that P does not equal NP.

Finally, we consider generalizations of OptP to higher levels in the Polynomial-Time Hierarchy. We define the DOUBLE KNAPSACK problem and prove that it is complete for DELTA-3, the first example of a natural complete problem for this class, and the highest level in the Polynomial Hierarchy with a known natural complete problem.

Biographical Sketch

Mark W. Krentel was born December 5, 1957 in Syracuse, New York and was raised in the beautiful country of central New York. He graduated from C. W. Baker High School in Baldwinsville, New York in June 1975 and enrolled at Clarkson College to study mathematics and computer science. He received the B.S. degree with great distinction in May 1978 and the M.S. degree in August 1979. He then enrolled in the graduate school at Cornell University in the fall of 1979 to study computer science. An unusually long "post-A-exam slump," combined with an irrational fear of the "real world" kept him in graduate school longer than he had expected. He will receive the Ph.D. degree in May 1987 and begins his academic career with a one-year position at the University of Chicago, followed by a regular appointment at Rice University in the fall of 1987. He shares the record for the most number of months playing tennis outdoors in Ithaca in a single calendar year at twelve.

Acknowledgements

First and foremost, I want to thank my parents. Right after them come Vijay Vazirani, Fred Schneider and Richard Shore for serving on my committee. Also, thanks go to Donald Knuth and Leslie Lamport for the TEX and LATEX systems.

Three classes, one in high school, one in college and one in graduate school, stand out in my memory. Freshman English, Topology and Algorithms, taught by Barbara Gamage, Charlie Marshall and John Hopcroft, were all quantum leaps forward for me. I worked harder in these classes than I've ever worked before or since, and I got as much out of them as I put in. The latter two were also great leaps forward in my level of mathematical sophistication and brought out new powers in me that I was previously unaware of. Thank you.

I also want to thank Cynthia Dwork, Debbie Joseph and Vijay Vazirani for some important pushes at key moments, David Shmoys for excellent support and assistance during some very stressful times, and also John

Hopcroft and Chris Buckley for refusing to let me quit when I wanted to. It is perhaps only a slight exaggeration to say that I would never have graduated without them.

Just as important are all the people who make life enjoyable. Betsy, Bowen, Carl, Chris, Cynthia, Doug, Ellen, Hans, Jim, Joe, John, Mark, Maura, Vivian and many others come to mind. There are also the bridge players and the softball and volleyball teams. Lastly, but certainly not least, are Maxine and Phoebe, who are just cats and will probably never know that they are being acknowledged here; but what the heck, they're real good cats and I love them very much.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is about optimization problems, and in particular, about NP-complete optimization problems. We intend to study these problems at a deeper level of structure and show that they indeed possess a very rich structure.

The goal of complexity theory is to classify the difficulty of natural problems. Typically, you pick some interesting resource, such as the amount of time or space needed for some computation, and then you define complexity classes parameterized by the amount of this resource. That leaves two major jobs for the complexity theorist. The first is to precisely characterize the amount of this resource that various problems require. Ideally, of course, we strive for matching upper and lower bounds, but this is, in general, a very difficult task. The other job is to prove hierarchy theorems for the

complexity classes. That is, show that given more of the resource allows us to compute strictly more functions. This too is usually very difficult.

In this thesis, we focus on the important class of NP problems. A language is in NP if it can be decided by a nondeterministic polynomially time bounded Turing machine. For example, the CLIQUE problem (given a graph G and integer k, are there k mutually adjacent vertices in G?) can be solved by such a machine by guessing the set of vertices and then verifying that the set has the correct size and that it forms a clique. NP is contrasted with the class P, the class of languages decidable in polynomial time by a deterministic Turing machine. P formalizes the notion of what is feasibly *computable*, while NP formalizes the notion of what is feasibly *verifiable*. The P =? NP question, currently the major open question in theoretical computer science, boils down to whether or not the ability to (magically) guess allows one to compute things faster. It goes without saying that this is also a very difficult problem.

Since the problems of separating complexity classes and of proving lower bounds are so difficult, complexity theorists often turn to completeness results. A language L_0 is *complete* for a class $\mathcal{C}$ if $L_0 \in \mathcal{C}$ and if all other languages $L \in \mathcal{C}$ can be reduced to L_0. For example, given a graph G_1, it is possible to construct (in polynomial time) a graph G_2 and an integer k such that G_1 contains a Hamilton cycle if and only if G_2 contains a clique

of size k. This shows that HAMILTON CYCLE is *reducible* to CLIQUE. The significance of a completeness result for NP is that although we don't know how to prove that CLIQUE $\notin$ P (this would imply P $\neq$ NP), we can show that CLIQUE is NP-complete and hence is as hard as any NP language. This is the best characterization of the complexity of CLIQUE that is currently known.

The traditional approach to complexity theory is to first convert the problem in question to a yes/no decision procedure. For example, the TRAVELLING SALESPERSON problem (TSP) is, given a graph G with costs on the edges, find a cycle in G that visits every node exactly once and minimizes the length of the cycle. This problem is converted to the question, given a graph G with costs on the edges and an integer k, does G have a TSP tour of cost at most k? And because we could solve the original problem by using this modified version as a subroutine, we say that this transformation captures the essential difficulty of the TSP problem.

The main focus of this thesis is to consider NP-complete problems at a deeper level of structure. In order to do this, we need to study the original versions of these problems and not convert them to yes/no questions. We define OptP to be the class of functions computable by taking the maximum (or minimum) value of some function over a set of feasible solutions, and we define OptP$[z(n)]$ to be the subclass of OptP containing those functions

restricted to $z(n)$ bits of output. The size of the largest clique in a graph, for example, can be computed in this manner. Consider any set of vertices to be a feasible solution, and assign the measure of the number of nodes in the set if it forms a clique and 0 otherwise. Valiant [Va79] used an analogous approach in defining the class of functions #P by taking the *sum* of the values of some function over a set of feasible solutions.

This approach has several advantages. The first is that it enables us to show more structure in NP-complete problems than was previously known. In chapter 2 we show that TRAVELLING SALESPERSON, WEIGHTED SATISFIABILITY and KNAPSACK are complete for OptP under a generalized notion of reducibility that we call the *metric reduction.* The techniques used in these reductions generalize the NP-completeness proofs for these problems and show that one problem can simulate much more of the structure of another problem than just its yes/no question. Skiena [Sk85] defines a Solitaire game Turing machine and the complexity classes SGP and SG'P, identical to our notions of OptP and OptP[$O(\log n)$]. He also independently proves several problems complete for SGP and SG'P, including many of the problems we give in chapter 2.

In chapter 3 we define $\mathsf{FP}^{\mathrm{SAT}}[z(n)]$, the class of functions computable in polynomial time with $z(n)$ queries to an NP oracle. It is immediate that $\mathsf{FP}^{\mathrm{SAT}}[z(n)]$ contains OptP[$z(n)$] by using the NP oracle to conduct

a binary search on the value of the OptP function. We prove as our main result that any $f \in \mathsf{FP}^{\mathrm{SAT}}[z(n)]$ can be reduced to some $g \in \mathsf{OptP}[z(n)]$. This shows that the parameter on OptP corresponds to the "amount" of NP-completeness in the problem. For example, the answers to $O(\log n)$ yes/no NP questions can be encoded into a single instance of CLIQUE. Similarly, the optimal value of a TSP problem contains the answers to $O(n)$ NP questions. Gasarch [Ga86] also considers the number of queries it takes to compute various functions and defines the class $Q[z(n), \mathsf{NPC}]$, identical to our notion of $\mathsf{FP}^{\mathrm{SAT}}[z(n)]$. He independently proves hardness results for many NP functions, but unfortunately, his lower bound techniques don't allow for successive queries to depend on the answers to the previous ones and thus he ends up with somewhat weaker bounds.

This result also allows us to tie together the classes NP, D^p and Δ_2^p. We show that under very general conditions, an OptP-completeness result also yields NP-, D^p- and Δ_2^p-completeness results. Although it was previously known that different versions of some problem could give different completeness results, it was not known how to prove a general result that tied these classes together. For example, for the TRAVELLING SALESPERSON problem, the question "Is the optimal value at most k?" is NP-complete [Ka72], the question "Is the optimal value equal to k?" is D^p-complete [PY84], and the question "Is the optimal solution unique?" is Δ_2^p-complete [Pa84]. In

chapter 3 we show that these results for NP and D^p and a similar result for Δ_2^p can be obtained as corollaries of the single OptP-completeness result for TSP.

In chapter 4 we consider separation results among $\mathsf{FP}^{\mathrm{SAT}}$ classes. Assuming $\mathsf{P} \neq \mathsf{NP}$, we show that $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$ is strictly contained in $\mathsf{FP}^{\mathrm{SAT}}[O(n)]$ and also that $\mathsf{FP}^{\mathrm{SAT}}[f(n)]$ is strictly contained in $\mathsf{FP}^{\mathrm{SAT}}[g(n)]$ for "sufficiently nice" f and g whenever $f(n) < g(n)$ and $f(n) \leq \frac{1}{2}\log n$. Since TSP is complete for $\mathsf{FP}^{\mathrm{SAT}}[O(n)]$ and since CLIQUE is complete for $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$, this result shows that TSP is strictly harder than CLIQUE in this measure. Karmarkar and Karp [KK82] showed that BIN PACKING can be approximated to within an additive constant of at most $O(\log^2 n)$. This implies that BIN PACKING is in $\mathsf{FP}^{\mathrm{SAT}}[O(\log\log n)]$ and hence that CLIQUE is strictly harder than BIN PACKING. Of course, all of these problems, considered as yes/no questions, are equivalent — they are all NP-complete. Our approach allows us to make finer distinctions on their complexity.

And finally, in chapter 5, we consider extensions of OptP to the polynomial hierarchy. We give a complete problem for the second level in this hierarchy and show that it naturally yields a complete problem for Δ_3^p. This is the first example of a complete problem for Δ_3^p and is also the highest level in the polynomial hierarchy with a known natural complete problem.

Chapter 2

Optimization Problems

2.1 Introduction

In this section, we first give some preliminary notation, and then we define the class OptP and some related notions.

Notation We write $\mathbf{N} = \{0, 1, 2, \ldots\}$ for the set of natural numbers and $\mathbf{Q}$ for the set of rationals.

First, we define metric Turing machines and the class OptP in order to capture our intuitive notion of an optimization problem.

Definition An NP *metric Turing Machine*, N, is a non-deterministic polynomially time-bounded Turing machine such that every branch writes

a binary number and accepts; and for $x \in \Sigma^*$ we write $\mathbf{opt}^N(x)$ for the largest value (for a maximization problem) on any branch of N on input x.

Definition A function $f: \Sigma^* \to \mathbf{N}$ is in OptP (optimization polynomial time) if there is an NP metric Turing machine N such that $f(x) = \mathbf{opt}^N(x)$ for all $x \in \Sigma^*$. We say that f is in OptP$[z(n)]$ if $f \in$ OptP and the length of $f(x)$ in binary is bounded by $z(|x|)$ for all $x \in \Sigma^*$.

Note that OptP is the same as OptP$[n^{O(1)}]$. Also note that OptP is defined as a class of *maximization* problems. We could equally as well have used minimization problems, and although we will only define the formalism for maximization problems, we will consider OptP as including both maximization and minimization problems.

One motivation for the class OptP is its similarity to Valiant's class #P (sharp-P, or number-P) [Va79]. Valiant defined *counting Turing machines* to be NP machines that magically output the number of accepting branches, or equivalently, the sum of the values over all of the branches. Then, #P is the class of functions computable by counting Turing machines. Valiant goes on to show that #P is an interesting class of functions by showing that several natural problems, including the PERMANENT and NUMBER OF SATISFYING ASSIGNMENTS are complete for it. Thus it is natural to consider other associative operators such as the MAX function.

The natural notion of reducibility between OptP problems is the metric reduction, the obvious generalization of a many-one reduction.

Definition Let $f, g: \Sigma^* \to \mathbf{N}$. A *metric reduction from f to g* is a pair of polynomial-time computable functions (T_1, T_2) where $T_1: \Sigma^* \to \Sigma^*$ and $T_2: \Sigma^* \times \mathbf{N} \to \mathbf{N}$ such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Sigma^*$.

Note that we need a many-one reduction here. Eventually (see chapter 4), we will want to bring out distinctions such as saying that computing the size of the largest clique in a graph is harder than its corresponding yes/no question. If we used Turing reductions, for example, then these distinctions would be blurred. Also note that because a metric reduction can stretch its input by a polynomial amount, if f is complete (under metric reductions) for $\mathsf{OptP}[z(n)]$, then f is also complete for $\mathsf{OptP}[z(n^{O(1)})]$. And lastly, note that metric reductions are closed under composition.

We are now ready to show, for "sufficiently nice" bounds $z(n)$, that $\mathsf{OptP}[z(n)]$ has complete functions. We first show that LEX and the universal function, UNIV, are complete via generic reductions. In the next section, we show natural complete functions for OptP and $\mathsf{OptP}[O(\log n)]$.

Definition A function $z: \mathbf{N} \to \mathbf{N}$ is *smooth* if z is non-decreasing and if the function $1^n \mapsto 1^{z(n)}$ is computable in polynomial time.

Lemma *Let $z(n)$ be smooth. The following functions are complete for* $\mathsf{OptP}[z(n)]$.

- $\text{UNIV}_{z(n)}$

  instance: $N\#x\#0^k$ where N is an NP metric Turing machine.

  output: $\text{UNIV}_{z(n)}$ simulates $N(x)$ for k moves and outputs the same value; branches that do not halt within k steps or output more than $z(|x|)$ bits have value 0.

- $\text{LEX}_{z(n)}$

  instance: Boolean formula $\varphi(x_1, \ldots, x_n)$.

  output: The lexicographically maximum $x_1 \cdots x_{z(|\varphi|)}$ that can be extended to a satisfying assignment.

Proof: UNIV. By a generic reduction. Let $f \in \mathsf{OptP}[z(n)]$, let N be an NP metric TM that computes f, and let N run in time $p(n)$ for some polynomial p. Then, for $x \in \Sigma^*$, reduce x to $T_1(x) = N\#x\#0^{p(|x|)}$. By the definition of UNIV_z, we have

$$\mathbf{opt}^N(x) = \mathbf{opt}^{\text{UNIV}}(T_1(x)),$$

which gives us a metric reduction from N to UNIV_z. □

Proof: LEX. We reduce UNIV_z to LEX_z. Let $y = N\#x\#0^k$ be an instance of UNIV_z. By Cook's theorem [Co71], there is a 3CNF boolean formula $\varphi(x_1, \ldots, x_n)$, with $|\varphi|$ polynomial in $|y|$, which says that "$x_1 \cdots x_n$

encode a legal computation of some branch of $\text{UNIV}_z(y)$ and $x_1 \cdots x_{z(|\varphi|)}$ represent the output on this branch." Then,

$$\mathbf{opt}^{\text{UNIV}}(y) = \mathbf{opt}^{\text{LEX}}(\varphi),$$

so we have a metric reduction from UNIV_z to LEX_z. □

2.2 OptP-Completeness Results

In this section, we give the reductions to show that WEIGHTED SATISFIABILITY, TRAVELLING SALESPERSON, MAXIMUM SATISFYING ASSIGNMENT, 0-1 INTEGER LINEAR PROGRAMMING and KNAPSACK are all complete for OptP. Of course, these problems (converted to decision procedures) were all known to be NP-complete. The reductions given here, in addition to showing that these problems are NP-complete, also show how to embed extra structure in them.

Theorem 2.1 *The following functions are complete for* OptP *under metric reductions.*

- WEIGHTED SATISFIABILITY

  instance: CNF boolean formula with (binary) weights on the clauses.

  output: The maximum weight of any assignment, where the weight of an assignment is the sum of the weights on the true clauses.

- TRAVELLING SALESPERSON

  instance: Weighted graph G.

  output: The length of the shortest travelling salesperson tour in G.

- MAXIMUM SATISFYING ASSIGNMENT

  instance: Boolean formula $\varphi(x_1, \ldots, x_n)$.

  output: The lexicographically maximum $x_1 \cdots x_n \in \{0,1\}^n$ that satisfies φ.

- 0-1 INTEGER LINEAR PROGRAMMING

  instance: Integer matrix A and vectors B and C.

  output: The maximum value of CX over all 0-1 vectors X subject to $AX \leq B$.

- KNAPSACK

  instance: Integers $x_1, \ldots, x_n, N$.

  output: The largest value, less than N, of $\sum_{i \in S} x_i$ taken over all $S \subseteq \{1, \ldots, n\}$.

Proof: WEIGHTED SATISFIABILITY. We reduce UNIV_n to WEIGHTED SAT. For $x \in \Sigma^*$, let $n = |x|$ and define the boolean formula $\varphi_x(x_1, \ldots, x_m, y_1, \ldots, y_n)$ to mean "$x_1, \ldots, x_m$ encode a legal computation of some branch of $\text{UNIV}_n(x)$; and $y_1 \cdots y_n$ is the binary representation of the output on this branch." Clearly, φ_x can be verified in polynomial time; therefore, by

Cook's theorem [Co71], we can encode φ_x as a CNF formula where m is polynomial in n, the length of x.

Reduce x to the CNF formula $\Phi_x = (\varphi_x)^{2^{2n}}(y_1)^{2^{n-1}}(y_2)^{2^{n-2}} \cdots (y_n)^1$. We use the notation $(\psi)^w$ to mean that all of the clauses in ψ have weight w. Clearly, φ_x is satisfiable, since any branch of UNIV_n will give a valid computation; therefore, the optimal assignment to Φ_x must satisfy φ_x. This means that the maximum number of simultaneously satisfiable clauses in Φ_x must be equal to the optimal value of UNIV_n on x plus 2^{2n} times the number of clauses in φ_x. That is,

$$\mathbf{opt}^{\text{W. SAT}}(\Phi_x) = \mathbf{opt}^{\text{UNIV}_n}(x) + \text{const}_x.$$

This gives a metric reduction from UNIV_n to WEIGHTED SATISFIABILITY, and hence WEIGHTED SAT is complete for OptP. □

Proof: TRAVELLING SALESPERSON. We reduce WEIGHTED SAT to TSP. The reduction is in two parts: first we reduce WEIGHTED SAT to CONSTRAINED TSP and then we remove the constraints. Papadimitriou used the same technique to show that the problem of deciding if an instance of TSP has a unique optimum solution is Δ_2^p complete [Pa84].

Suppose φ is an instance of WEIGHTED SAT with variables $x_1, \ldots, x_n$, and clauses $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m$. Reduce φ to the weighted graph $G = (V, E, c)$ as follows. G is basically a large cycle with occasional multiple edges to represent choices for the variables and clauses in φ. For

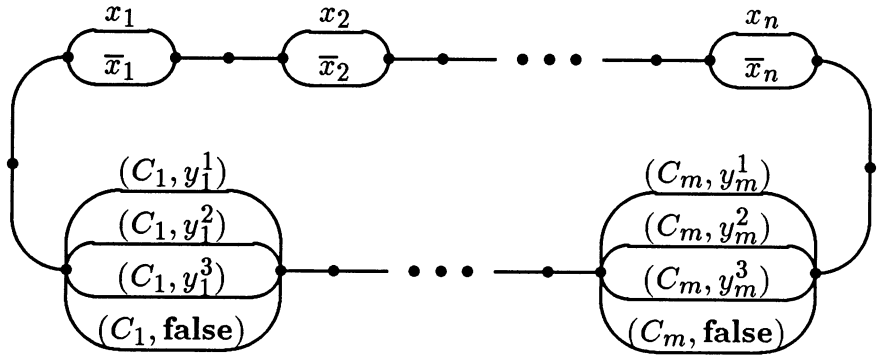

Figure 2.1: TSP construction.

---

each variable x_i, include a pair of nodes and two edges between them: one labelled x_i and one labelled $\overline{x}_i$. It will turn out that any tour in G will use either the x_i or the $\overline{x}_i$ edge; this choice represents a truth assignment to x_i. For each clause C_j containing literals y_j^1, y_j^2, and y_j^3, include a pair of nodes and four edges between them: three with labels (C_j, y_j^1), (C_j, y_j^2) and (C_j, y_j^3) and one labelled $(C_j, \textbf{false})$. Again, any tour in G will use exactly one of these edges; this choice corresponds to a true literal, if any, in C_j. Connect these nodes and edges in a large cycle as in figure 2.1. Although the graph has multiple edges, we will later put constraints on the set of allowed tours; replacing these constraints will remove the multiple edges.

The cost of edge $(C_j, \textbf{false})$ is w_j, the weight on clause C_j. The cost of every other edge is 0, and the cost of the non-edges is $+\infty$. We disallow

tours that represent illegal assignments by the following constraints. A NAND constraint between edges e_1 and e_2 specifies that a tour is not allowed to use both e_1 and e_2. For each literal x_i and each occurrence of $\overline{x}_i$ in clause C_j, include a NAND constraint between x_i and $(C_j, \overline{x}_i)$. Also include a NAND constraint for each pair $\overline{x}_i$ and (C_j, x_i).

A tour that obeys the NAND constraints represents a legal truth assignment to the variables. Also, a tour can use the edge (C_j, y) only if y is set to true. Thus the cost of a tour, the sum of the weights on the **false** edges, is also the sum of the weights of the unsatisfied clauses in φ. (Although a tour may use the **false** edge of a true clause, this can't happen in an optimal tour.) Thus,

$$\mathbf{opt}^{\text{W. SAT}}(\varphi) = \text{const}_\varphi - \mathbf{opt}^{\text{TSP}}(G).$$

This gives a metric reduction from WEIGHTED SAT to CONSTRAINED TSP.

The reduction from CONSTRAINED TSP to TSP uses a 108-node gadget to implement the NAND constraints. This gadget, a combination of two other gadgets, is described and proved correct in [Pa84]. □

Proof: MAXIMUM SATISFYING ASSIGNMENT. The reduction from UNIV to WEIGHTED SAT also works here. Reduce x to φ_x and order the variables $y_1, \ldots, y_n, x_1, \ldots, x_m$. Then, $\mathbf{opt}^{\text{UNIV}_n}(x)$ can be found from the high-order bits of $\mathbf{opt}^{\text{MAX SAT ASSGN}}(\varphi_x)$. □

Proof: 0-1 INTEGER LINEAR PROGRAMMING. We reduce WEIGHTED SAT to 01ILP. Let φ be a weighted CNF formula with variables $x_1, \ldots, x_n$, clauses $C_1, \ldots, C_m$ and weights $w_1, \ldots, w_m$. Reduce φ to an instance I of 01ILP with variables $x_1, \ldots, x_n$, $\overline{x}_1, \ldots, \overline{x}_n$, and $c_1, \ldots, c_m$. For each variable x_i, include the constraint $x_i + \overline{x}_i = 1$, and for each clause $C_j = (y_1 + y_2 + y_3)$, include the constraint $y_1 + y_2 + y_3 - c_j \geq 0$. Then, a 0-1 solution to this problem represents a legal truth assignment to the variables in φ, and c_j can be set to 1 only if at least one of the literals in clause C_j is set to true. Thus, by using $c_1 w_1 + \cdots + c_m w_m$ as the objective function, we see that

$$\mathbf{opt}^{\text{W. SAT}}(\varphi) = \mathbf{opt}^{\text{01ILP}}(I).$$

Thus, 0-1 INTEGER LINEAR PROGRAMMING is OptP complete. □

Proof: KNAPSACK. We reduce MAXIMUM SATISFYING ASSIGNMENT to KNAPSACK. In order to simplify the construction, we use a standard trick in KNAPSACK-style reductions, as in [GJ79]. We write numbers in base r, where r is a sufficiently large number to be chosen later. The idea is that r will be large enough so that the bits of a base r number will represent independent "zones." Thus, we may assume that there are no possibilities of carries in the numbers that we use.

Let φ be a CNF boolean formula with variables x_1, ..., x_n and clauses C_1, ..., C_m. We reduce φ to K, an instance of KNAPSACK. Altogether we

Table 2.1: OptP KNAPSACK construction.

	x_1^1	x_i^1	x_n^1	C_1^1	C_j^1	C_m^1	C_1^2	C_j^2	C_m^2	x_1^2	x_i^2	x_n^2
x_i		1		1 for each clause x_i is in							1	
$\overline{x}_i$		1		1 for each clause $\overline{x}_i$ is in							0	
C_j^0					0			1				
C_j^1					1			1				
C_j^2					2			1				
N	1	$\cdots$	1	3	$\cdots$	3	1	$\cdots$	1	1	$\cdots$	1
M	1	$\cdots$	1	3	$\cdots$	3	1	$\cdots$	1	0	$\cdots$	0

use $2n + 2m$ zones in four categories: for each variable x_i, make zones x_i^1 and x_i^2; and for each clause C_j, make zones C_j^1 and C_j^2. We also use $2n+3m$ numbers in five categories: for each variable x_i, make integers x_i and $\overline{x}_i$; and for each clause C_j, make integers C_j^0, C_j^1 and C_j^2. The construction of K is summarized in table 2.1. Intuitively, the x_i^1 zone guarantees that x_i is set to either true or false but not both. The x_i^2 zone is used to weight $x_i =$ **true** heavier than $x_i =$ **false**; thus the weight of an assignment to $x_1 \cdots x_n$ corresponds to its lexicographic order. The C_j^1 and C_j^2 zones are used to verify that the asignment to $x_1 \cdots x_n$ satisfies clause C_j.

Now we can choose the value for r. Since the largest digit in any column is 2 (N and M don't count), and since there are $2n + 3m$ numbers, setting $r = 2(2n + 3m) + 1$ will guarantee that there are no possibilities of any carries.

Thus we see that a solution to K that sums to a value larger than M must correspond to a satisfying assignment. Conversely, given a satisfying assignment to φ, we can find a solution to K with value at least M. Thus the optimal value for K is greater than M if and only if φ has a satisfying assignment. Furthermore, by weighting $x_i =$ **true** heavier than $x_i =$ **false** in the x_i^2 zone, we see that

$$\mathbf{opt}^{\text{KNAPSACK}}(K) = \mathbf{opt}^{\text{MAX SAT ASSGN}}(\varphi) + M.$$

Thus, KNAPSACK is OptP complete. □

2.3 OptP[O(log n)]-Completeness Results

In this section, we give the reductions to show that MAX SAT, CLIQUE, COLORING and LONGEST CYCLE are all complete for OptP[$O(\log n)$]. Of special interest is the reduction for COLORING. Karp's reduction for k-COLORING [Ka72] constructs a graph that is k-colorable if a given boolean formula is satisfiable and $(k + 1)$-colorable otherwise. Similarly, the NP-completeness proof for 3-COLORING [GJS76] constructs a graph with chro-

matic number 3 or 4. These reductions, put in our framework, would only show that COLORING is hard for OptP[1]. Our construction shows that there is much more information in the chromatic number of a graph than just the answer to a single yes/no NP question.

Theorem 2.2 *The following functions are complete for* OptP[$O(\log n)$] *under metric reductions.*

- MAXIMUM SATISFIABILITY

  instance: CNF formula φ.

  output: The maximum number of simultaneously satisfiable clauses.

- CLIQUE

  instance: Graph G.

  output: The size of the largest clique in G.

- COLORING

  instance: Graph G.

  output: The chromatic number of G.

- LONGEST CYCLE

  instance: Graph G.

  output: The length of the longest cycle in G.

Proof: MAXIMUM SATISFIABILITY. The proof for WEIGHTED SAT can be modified to give a reduction from $\text{UNIV}_{\log n}$ to MAX SAT. We can remove

the weights by repeating clauses, since the weights for $\text{UNIV}_{\log n}$ only need to be polynomially large. □

Proof: MAXIMUM CLIQUE. The reduction from SAT to CLIQUE given in [Ka72] or in [AHU74] has the property that the size of the maximum clique is equal to the maximum number of simultaneously satisfiable clauses. □

Proof: COLORING. First we show how to reduce a boolean formula φ to a graph G_n such that $\chi(G) = n$ if $\varphi \in$ SAT and $\chi(G) = 2n - 4$ if $\varphi \notin$ SAT, and then we use this result to show that MAX SAT is metrically reducible to COLORING. In order to make the graph G_n, we need the idea of a multicoloring. A (k, m)-*multicoloring* of a graph $G = (V, E)$ is a function f that assigns to each $v \in V$ a set $f(v) \subseteq \{1, \ldots, m\}$ such that $|f(v)| = k$ and such that $f(u)$ and $f(v)$ are disjoint whenever $(u, v) \in E$. We write $\chi_k(G)$, the *k-chromatic number of* G, for the smallest integer m such that G has a (k, m)-multicoloring.

Let φ be a boolean formula, and pick an integer n. By the "True-False-Red" reduction of SAT to 3-COLORING in [GJS76] and in [AHU74], we can make a graph G such that $\chi(G) = 3$ if $\varphi \in$ SAT and $\chi(G) = 4$ if $\varphi \notin$ SAT. Now, define the graph $H_n = (V_n, E_n)$ where

$$\begin{aligned} V_n &= \{\{i, j, k\} \mid 1 \leq i < j < k \leq n\} \\ E_n &= \{(u, v) \mid u \wedge v = \emptyset\}. \end{aligned}$$

The key property of these graphs, described in [GJ79] and proved in [GJ76], is that $\chi_3(H_n) = n$ and $\chi_4(H_n) = 2n-4$. It is straightforward to verify that if $\chi(G) = k$ and if $\chi_k(H) = m$ then $\chi(H[G]) = m$. So, let $G_n = H_n[G]$, the composition of H_n and G, and thus,

$$\chi(G_n) = \begin{cases} n & \text{if } \varphi \text{ is satisfiable} \\ 2n-4 & \text{if } \varphi \text{ is not satisfiable.} \end{cases}$$

Now let Φ be a CNF formula with n clauses. From Cook's theorem, we can construct boolean formulas $\varphi_1, \ldots, \varphi_{n+1}$ such that φ_i is satisfiable if and only if Φ has an assignment of its variables that satisfies at least i clauses. By the previous paragraph, we can construct graphs $G_1, \ldots, G_{n+1}$ such that

$$\chi(G_i) = \begin{cases} 2n-i & \text{if } \varphi_i \text{ is satisfiable} \\ 4n-2i-4 & \text{if } \varphi_i \text{ is not satisfiable.} \end{cases}$$

Define $G^* = G_1 + \cdots + G_{n+1}$, the disjoint union of the G_i's, so that $\chi(G^*) = \max_i \chi(G_i)$, and let $k = \mathbf{opt}^{\text{SAT}}(\Phi)$. Then, $\varphi_1, \ldots, \varphi_k$ are satisfiable but $\varphi_{k+1}, \ldots, \varphi_{n+1}$ are not, so $\chi(G^*) = \chi(G_{k+1}) = 4n - 2k - 6$ and hence

$$\mathbf{opt}^{\text{COLORING}}(G^*) = 4n - 6 - 2\mathbf{opt}^{\text{MAX SAT}}(\Phi).$$

Thus, COLORING is OptP[$O(\log n)$] complete. □

Proof: LONGEST CYCLE. The reduction from WEIGHTED SAT to TRAVELLING SALESPERSON can be modified to give a reduction from MAX SAT to

TSP such that the weights on the edges are only polynomially large. Then, we can remove the weights by repeating edges. □

Chapter 3

$\mathbf{P}^{\mathbf{NP}}$ Computations

3.1 Introduction

In this chapter, we consider functions computable in polynomial time with an oracle for NP, and we show that they are closely related to OptP functions. The main point of this thesis is that there is more structure in NP-complete problems than was previously known. By counting, as a complexity measure, the number of NP queries it takes to compute a function, we have a precise way of measuring "how much" NP-completeness is in a problem. It will turn out that, in this measure, some problems have more NP-completeness in them than others.

Definition A function $f: \Sigma^* \rightarrow \mathbf{N}$ is in $\mathsf{FP}^{\mathrm{SAT}}$ if f is computable in polynomial time with an oracle for NP. We say that f is in $\mathsf{FP}^{\mathrm{SAT}}[z(n)]$

if $f \in \mathsf{FP}^{\mathrm{SAT}}$ and f is computable using at most $z(n)$ queries on inputs of length n.

Definition A language $L \subseteq \Sigma^*$ is in $\mathsf{P}^{\mathrm{SAT}}$ if L is decidable in polynomial time with an oracle for NP. We say that L is in $\mathsf{P}^{\mathrm{SAT}}[z(n)]$ if $L \in \mathsf{P}^{\mathrm{SAT}}$ and L is computable using at most $z(n)$ queries on inputs of length n.

Note again that $\mathsf{FP}^{\mathrm{SAT}} = \mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$. Our main result is that an $\mathsf{FP}^{\mathrm{SAT}}$ function metrically reduces to an OptP function. This result says that the parameter on OptP makes sense as a complexity measure. There could be functions that are easy to compute that just have long outputs; we don't want to call them "hard." This result says that an $\mathsf{OptP}[f(n)]$-complete function is also complete for $\mathsf{FP}^{\mathrm{SAT}}[f(n)]$, and thus the parameter on OptP *does* make sense as a complexity measure. Thus, TRAVELLING SALESPERSON, WEIGHTED SAT, MAXIMUM SATISFYING ASSIGNMENT, 0-1 INTEGER LINEAR PROGRAMMING and KNAPSACK are all complete for $\mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$, and MAX SAT, CLIQUE, COLORING and LONGEST CYCLE are complete for $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$. This gives a precise characterization of "how much" NP-completeness these problems contain.

3.2 Main Result

First we show that every function in $\mathsf{FP}^{\mathrm{SAT}}$ decomposes into an OptP problem followed by a polynomial-time computation. The difficulty in the proof is in showing that an NP machine with the MAX function is as powerful as a $\mathsf{P}^{\mathrm{SAT}}$ machine. An NP machine could guess the entire $\mathsf{P}^{\mathrm{SAT}}$ computation and could even verify the 'yes' answers, but it has no way of verifying the 'no' answers. We get around this difficulty by trying all possible sequences of oracle answers and taking the maximum sequence for which all of the 'yes' answers are correct. In this way, the output of the OptP function represents the true oracle answers in the $\mathsf{P}^{\mathrm{SAT}}$ computation.

Theorem 3.1 *Let z be smooth. Then, any $f \in \mathsf{FP}^{\mathrm{SAT}}[z(n)]$ can be written as $f(x) = h(x, g(x))$ where $g \in \mathsf{OptP}[z(n)]$ and $h\colon \Sigma^* \times \mathbf{N} \to \mathbf{N}$ is computable in polynomial time.*

Proof: Let $f \in \mathsf{FP}^{\mathrm{SAT}}[z(n)]$, and let M compute f, where M is a $\mathsf{P}^{\mathrm{SAT}}$ machine making $z(n)$ queries on inputs of length n. Except for the answers to its queries to SAT, M's computation is in polynomial time; so, on input $|x| = n$, and given $b_1, \ldots, b_{z(n)} \in \{0, 1\}$, we can simulate M's computation in polynomial time, substituting $b_1 \cdots b_{z(n)}$ for the answers to M's queries.

We construct N, an NP metric Turing machine, as follows. On input $|x| = n$, N first computes $z(n)$ and then branches for each string

in $\{0,1\}^{z(n)}$. On branch $b_1 \cdots b_{z(n)}$, N simulates M and constructs M's queries on this branch, say, $\varphi_1, \ldots, \varphi_{z(n)}$. Then N tries to guess satisfying assignments for each φ_i such that $b_i = 1$ and ignores the φ_i's such that $b_i = 0$. If N successfully finds a satisfying assignment for each φ_i where $b_i = 1$, then N outputs the value $b_1 \cdots b_{z(n)}$ as a binary integer on this branch; otherwise N outputs 0.

Now, we claim that $\mathbf{opt}^N(x)$ represents the correct oracle answers for $M(x)$. Write $\mathbf{opt}^N(x)$ as $b_1 \cdots b_{z(n)} \in \{0,1\}^{z(n)}$. First, we show that b_1 is correct. Let φ_1 be M's first query. If $\varphi_1 \in$ SAT, then $N(x)$ on branch $10\cdots0$ will find a satisfying assignment; so, $\mathbf{opt}^N(x) \geq 10\cdots0$ and thus $b_1 = 1$. On the other hand, if $\varphi_1 \notin$ SAT, then no branch of the form $1d_2 \cdots d_{z(n)}$ for any $d_i \in \{0,1\}$ will find a satisfying assignment to φ_1; therefore, $\mathbf{opt}^N(x) \leq 01\cdots1$ and hence $b_1 = 0$. In either case, the value of b_1 is correct.

By the same argument, we see that b_2 is correct, given that b_1 is correct; and hence, by induction, all of the b_i's are the correct oracle answers in M's computation on x. And since we can run $M(x)$ in polynomial time given $\mathbf{opt}^N(x)$, we can write $f(x) = h(x, \mathbf{opt}^N(x))$ where h is computable in polynomial time. □

The converse of theorem 3.1 is immediate; therefore, we can completely characterize $\mathsf{P}^{\mathrm{SAT}}$ computations, for both functions and languages, in terms of OptP.

Theorem 3.2 *Let z be smooth.*

- *$f \in \mathsf{FP}^{\mathrm{SAT}}[z(n)]$ if and only if f can be written as $f(x) = h(x, g(x))$ where $g \in \mathsf{OptP}[z(n)]$ and h is p-computable.*
- *$f \in \mathsf{FP}^{\mathrm{SAT}}[z(n^{O(1)})]$ if and only if f is metrically reducible to some $g \in \mathsf{OptP}[z(n)]$.*
- *$L \in \mathsf{P}^{\mathrm{SAT}}[z(n)]$ if and only if L can be written as $L = \{x \mid P(x, g(x))\}$ where $g \in \mathsf{OptP}[z(n)]$ and P is a p-computable predicate.*

3.3 Applications

Our next result is that OptP complete problems give rise to complete problems for Δ_2^p, D^p and NP. We conjecture that if f is OptP-complete, then $\{x\#k \mid f(x) = k\}$ is D^p-complete. Unfortunately, the proof doesn't seem to go through directly: we need the additional hypothesis of a linear reduction.

Definition Let $f, g: \Sigma^* \to \mathbf{N}$. A *linear reduction* from f to g is a triple of p-computable functions (T_1, T_2, T_3) where $T_1: \Sigma^* \to \Sigma^*$, $T_2: \Sigma^* \to \mathbf{Q}\backslash\{0\}$, and $T_3: \Sigma^* \to \mathbf{Q}$, such that for all $x \in \Sigma^*$ we have $f(x) = T_2(x)g(T_1(x)) + T_3(x)$.

Thus, a linear reduction is a special case of a metric reduction where the function $k \mapsto T_2(x,k)$ is linear. Note, however, that the coefficients of the linear function may, in general, depend on the problem instance. All of the problems in section 2 are complete for their respective classes via linear reductions except for MAX SAT ASSIGNMENT. (Although the reduction for KNAPSACK is from MAX SAT ASSIGNMENT, the composition of reductions from WEIGHTED SAT to MAXIMUM SATISFYING ASSIGNMENT to KNAPSACK can be modified to give a linear reduction.)

Theorem 3.3 *Let f be in* OptP.

(i) If f is complete for OptP, *then there is a p-computable predicate P such that $L_1 = \{x\#y \mid P(x, f(y))\}$ is complete for Δ_2^p.*

(ii) If f is complete for OptP *via linear reductions, then $L_2 = \{x\#k_1\#k_2 \mid f(x) \equiv k_1 \,(\mathrm{mod} k_2)\}$ is complete for Δ_2^p.*

(iii) If f is hard for OptP[2] *via linear reductions, then $L_3 = \{x\#k \mid f(x) = k\}$ is complete for D^p.*

(iv) If f is hard for OptP[1] *via linear reductions, then $L_4 = \{x\#k \mid f(x) \geq k\}$ is complete for* NP.

Proof: (i) Let $L \in \Delta_2^p$ and let M be a Δ_2^p-machine recognizing L. Define $g(x)$ to be the sequence of correct oracle answers for M's computation on x. Certainly, $g \in \mathsf{FP}^{\mathrm{SAT}}$, and we are assuming that f is OptP-complete, so

by theorem 3.1, g is metrically reducible to f via (T_1, T_2). Then, $M(x)$'s oracle answers can be computed in polynomial time from x and $f(T_1(x))$. Construct $P(x, z)$ to simulate $M(x)$ using $T_2(x, z)$ as the answers for $M(x)$'s oracle questions. Thus, $x \in L$ if and only if $x\#T_1(x) \in L_1$, and thus L is reducible to L_1.

(ii) Suppose $L \in \mathsf{P}^{\mathrm{SAT}}$ and let M be a $\mathsf{P}^{\mathrm{SAT}}$ machine accepting L. Define $g(x)$ to be $M(x)$'s oracle answers followed by a 1 if M accepts or a 0 if M rejects. Then, $g \in \mathsf{OptP}$ by an argument similar to the proof of theorem 3.1. If g reduces to f via a linear reduction, then a question of the form, "Is $g(x) \equiv 1 \,(\mathrm{mod} 2)$?" reduces to a question of the form "Is $f(y) \equiv k_1 \,(\mathrm{mod} k_2)$?"

(iii) Define the function $g(x\#y) = 2$ if $y \in$ SAT; or 1 if $x \in$ SAT and $y \notin$ SAT; or 0 if $x, y \notin$ SAT. Then, $g \in \mathsf{OptP}[2]$. Also, the D^p complete problem SAT–UNSAT [PY84] can be expressed as $\{x\#y \mid g(x\#y) = 1\}$. If g reduces to f via a linear reduction, then a question of the form, "Is $g(x) = 1$?" reduces to a question of the form, "Is $f(y) = k$?"

(iv) Define the function $g(x) = 1$ if $x \in$ SAT, or 0 if $x \notin$ SAT. Then $g \in \mathsf{OptP}[1]$. If g reduces to f via a linear reduction, then a question of the form, "Is $g(x) \geq 1$?" reduces to a question of the form "Is $f(y) \geq k$?" □

We conclude this section by giving natural complete languages for $\mathsf{P}^{\mathrm{SAT}}$ and $\mathsf{P}^{\mathrm{SAT}}[O(\log n)]$. Previously, the only known example of a complete

language for $\mathsf{P}^{\mathrm{SAT}}$ was the UNIQUELY OPTIMAL TRAVELLING SALESPERSON problem [Pa84]. Kadin [Ka86] discusses $\mathsf{P}^{\mathrm{SAT}}[O(\log n)]$ and gives natural complete languages for this class.

Theorem 3.4 *The following languages are complete for* $\mathsf{P}^{\mathrm{SAT}}$.

- $\{\varphi(x_1, \ldots, x_n) \mid x_n = 1$ in φ's max sat assgn$\}$
- $\{G\#k \mid$ length of min TSP tour in G is equiv to 0 mod $k\}$

Theorem 3.5 *The following languages are complete for* $\mathsf{P}^{\mathrm{SAT}}[O(\log n)]$.

- $\{\varphi\#k \mid$ max number of simul sat clauses in φ is equiv to 0 mod $k\}$
- $\{G\#k \mid$ size of max clique in G is equiv to 0 mod $k\}$

Chapter 4

Separation Results

4.1 Introduction

In this chapter, we consider the question of which classes of OptP functions are provably distinct under the assumption that P $\neq$ NP, and we show that these results have applications to approximation algorithms for NP complete problems. Recall that one of the original motivations for considering problems as functions rather than as languages was to make finer distinctions on their complexity. We would like to say, for example, that since TRAVELLING SALESPERSON is complete for $\mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$, since CLIQUE is complete for $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$ and since BIN PACKING is in $\mathsf{FP}^{\mathrm{SAT}}[O(\log\log n)]$, that TSP is strictly harder than CLIQUE and that CLIQUE is strictly harder than BIN PACKING. It turns out that these three

classes are provably distinct, but only by considering them as functions. There are oracles where these problems, considered as decision procedures, are equivalent. In fact, there is an oracle for which $\mathsf{P}^{\mathrm{SAT}}$ collapses to just $\mathsf{P}^{\mathrm{SAT}}[1]$.

Lemma *There is an oracle A such that $\mathsf{P}^A \neq \mathsf{NP}^A$ and $\mathsf{P}^{\mathrm{SAT},A}[1] = \mathsf{P}^{\mathrm{SAT},A}$.*

Proof: Pick an oracle A such that $\mathsf{P}^A \neq \mathsf{NP}^A = \mathrm{co}\mathsf{NP}^A$ [BGS75]. Then, $\mathsf{NP}^A = \mathrm{co}\mathsf{NP}^A$ implies that $\mathsf{NP}^A = \mathsf{P}^{\mathrm{SAT},A}$ and hence $\mathsf{P}^{\mathrm{SAT},A}[1] = \mathsf{P}^{\mathrm{SAT},A}$. □

Thus, it is unlikely that current techniques are strong enough to answer this question for languages. On the other hand, the corresponding question for the optimization versions of the same problems *can* be resolved.

4.2 Separation Results

We prove two separation results: the first is that n queries are strictly more powerful than $O(\log n)$ queries. As a corollary, this result shows that there can be no metric reduction from TSP to CLIQUE, and hence TSP is strictly harder than CLIQUE.

Theorem 4.1 $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)] = \mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ *implies* $\mathsf{P} = \mathsf{NP}$.

Proof: Assume $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)] = \mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$. Then we show that $\mathsf{P} = \mathsf{NP}$ by showing how to recognize SATISFIABILITY in polynomial time. By hypothesis, LEX $\in \mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$, so there is a $\mathsf{P}^{\mathrm{SAT}}$ machine, M, that computes LEX and makes at most $O(\log n)$ queries. Then, to determine if $\varphi \in$ SAT, simulate $M(\varphi)$ for all possible oracle answers. This gives a polynomial number of possible assignments, at least one of which must be a satisfying assignment if $\varphi \in$ SAT. □

We also prove a more general separation result: $\mathsf{FP}^{\mathrm{SAT}}[f(n)]$ is properly contained in $\mathsf{FP}^{\mathrm{SAT}}[g(n)]$ whenever $f(n) < g(n)$ and $f(n) \leq \frac{1}{2}\log n$. A corollary of this result is that CLIQUE is harder than BIN PACKING. Karmarkar and Karp [KK82] show that BIN PACKING can be approximated in polynomial time within an additive constant of $O(\log^2 n)$. The exact optimal number of bins can then be found with only $2\log\log n + O(1)$ queries to SAT, and so BIN PACKING is in $\mathsf{FP}^{\mathrm{SAT}}[2\log\log n + O(1)]$. CLIQUE, on the other hand, is complete for $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$.

Theorem 4.2 *Let f and g be smooth where $f(n) < g(n)$ and $f(n) \leq \frac{1}{2}\log n$. Then, $\mathsf{FP}^{\mathrm{SAT}}[f(n)] = \mathsf{FP}^{\mathrm{SAT}}[g(n)]$ implies $\mathsf{P} = \mathsf{NP}$.*

Proof: Assume that $\mathsf{FP}^{\mathrm{SAT}}[f(n)] = \mathsf{FP}^{\mathrm{SAT}}[g(n)]$ where f and g are as above. By hypothesis, $\mathrm{LEX}_g \in \mathsf{FP}^{\mathrm{SAT}}[f(n)]$, so let M be a $\mathsf{P}^{\mathrm{SAT}}$ machine that computes LEX_g with only $f(|\varphi|)$ oracle queries.

To test for satisfiability, let $\varphi(x_1, \ldots, x_n)$ be a boolean formula of length at most n. Simulate $M(\varphi)$ for all possible oracle answers; we can do this because $f(n) < \log n$. Then, we can write $x_1, \ldots, x_{g(n)}$ as a function of $y_1, \ldots, y_{f(n)}$, the oracle answers for $M(\varphi)$. Express this relation, $T(x_1, \ldots, x_{g(n)}, y_1, \ldots, y_{f(n)})$, as a truth table of size $\leq g(n) \cdot 2^{f(n)} \cdot \log n$. Since we may as well assume that $g(n) = f(n) + 1$, this has size at most n. Rewrite φ as $\varphi' = \varphi \wedge T$ with the variables in the order $y_1, \ldots, y_{f(n)}$, $x_{g(n)+1}, \ldots, x_n, x_1, \ldots, x_{g(n)}$, and say that $y_1, \ldots, y_{f(n)}$ are "independent" and that $x_1, \ldots, x_{g(n)}$ are "dependent." Then, φ is satisfiable if and only if φ' is satisfiable. And since $f(n) < g(n)$, we have eliminated at least one independent variable in φ by increasing the length of φ by an additive amount of at most n.

We can repeat this process with input φ' to make a formula φ'' of length $3n$, and so on. Since we always eliminate at least one independent variable, we never need more than n iterations to remove all but $f(n)$ of the independent variables. Then, we can try all possible values for the remaining $f(n)$ independent variables. Since the formula never grows beyond size n^2, we can solve SAT in polynomial time. □

4.3 Applications

This last separation result has applications to approximation algorithms for NP complete problems. If Π is an $\mathsf{OptP}[f(n)]$ complete problem, then embedded in the optimal value of Π are the answers to $f(n)$ NP complete questions. This gives a lower bound on how closely Π can be approximated. If we can approximate Π within an additive constant less than $2^{f(n)}$, then we can find the exact optimal solution with fewer than $f(n)$ queries, a contradiction to theorem 4.2. The result is a lower bound of $\frac{1}{2}f(n^\epsilon)$ rather than $f(n)$ because the reduction from LEX_f to Π might stretch the input by a polynomial amount.

Theorem 4.3 *Suppose* Π *is* $\mathsf{OptP}[f(n)]$ *complete, where* $f \in O(\log n)$ *is smooth, and suppose* $\mathsf{P} \neq \mathsf{NP}$. *Then, there exists an* $\epsilon > 0$ *such that any polynomial-time approximation algorithm* A *for* Π *must have* $|A(I) - \mathbf{opt}(I)| \geq \frac{1}{2}2^{f(n^\epsilon)}$ *infinitely often.*

Proof: As a corollary to the proof of theorem 4.2, $\text{LEX}_f \notin \mathsf{FP}^{\text{SAT}}[f(n) - 1]$, unless $\mathsf{P} = \mathsf{NP}$. Now, if Π is hard for $\mathsf{OptP}[f(n)]$, then there exists a metric reduction from LEX_f to Π. Since the reduction runs in polynomial time, it can only stretch the input by at most n^k for some k. Then, for $\epsilon < 1/k$, if a polynomial-time algorithm A could approximate Π within an additive constant of $\frac{1}{2}2^{f(n^\epsilon)}$, then we could solve Π with $f(n^\epsilon) - 1$ queries,

and hence we could solve LEX_f with only $f(n^{\epsilon k}) - 1$ queries. This is a contradiction to theorem 4.2 unless $\mathsf{P} = \mathsf{NP}$. $\square$

Chapter 5

Polynomial-Time Hierarchy

5.1 Introduction

The main goal of this chapter is to show that OptP has a natural generalization to higher levels in the polynomial-time hierarchy, and, as a corollary, to give a natural complete problem for Δ_3^p. The polynomial-time hierarchy was defined by Stockmeyer [St77,CKS81] as a generalization of the class NP as follows.

Definition The *polynomial-time hierarchy* consists of the classes Σ_k^p, Π_k^p and Δ_k^p, for each $k \geq 0$. Σ_k^p is the set of languages decidable by a polynomially time-bounded alternating Turing machine with k alternations of the AND and OR functions. Π_k^p is the set of languages whose complements are in Σ_k^p. And Δ_k^p is P with an oracle for Σ_k^p.

Thus, $\mathsf{NP} = \Sigma_1^p$. Just as Σ_k^p is a generalization of NP by considering alternating AND's and OR's, it is natural to consider alternating MAX and MIN functions. It turns out that this extension does indeed make sense and defines an interesting class of functions.

Definition A function $f: \Sigma^* \to \mathbf{N}$ is in Σ_k^{MM} if f is computable by a polynomially time-bounded alternating Turing machine with k alternations of the MAX and MIN functions.

Definition A function $f: \Sigma^* \to \mathbf{N}$ is in $\mathsf{F}\Delta_k^p$ if f is computable in polynomial time with an oracle for Σ_{k-1}^p in the Polynomial Time Hierarchy.

Thus, $\mathsf{OptP} = \Sigma_1^{MM}$ and $\mathsf{FP}^{\mathrm{SAT}} = \mathsf{F}\Delta_2^p$. The motivation for Σ_k^{MM}, besides being the natural generalization of OptP, is that it has applications to game theory. For example, Σ_k^{MM} is the natural class for computing the value of two-player, zero-sum games of perfect information with k moves and polynomial-time definable rules. The class $\mathsf{F}\Delta_k^p$ is the natural functional analogue of Δ_k^p in the polynomial hierarchy.

5.2 Equivalence Theorem

Our first result is that an $\mathsf{F}\Delta_k^p$ function decomposes into a Σ_k^{MM} function and a polynomial-time computation, in the same way that an $\mathsf{FP}^{\mathrm{SAT}}$

function decomposes into an OptP problem and a polynomial-time computation. The proof is essentially the same as the proof of theorem 3.1, just with more quantifiers.

Theorem 5.1 *Any* $f \in \mathsf{F}\Delta_{k+1}^p$ *can be expressed as* $f(x) = h(x, g(x))$ *where* $g \in \Sigma_k^{MM}$ *and* $h\colon \Sigma^* \times \mathbf{N} \to \mathbf{N}$ *is computable in polynomial time.*

Corollary *Let* $f\colon \Sigma^* \to \mathbf{N}$. *Then* $f \in \mathsf{F}\Delta_{k+1}^p$ *if and only if* f *is metrically reducible to some* $g \in \Sigma_k^{MM}$.

Proof: Let $f \in \mathsf{F}\Delta_{k+1}^p$ and let M be a p-machine with an oracle for Σ_k^p that computes f. Define $g(x) =$ the sequence of oracle answers, interpreted as a binary integer, in M's computation on x. Since we can easily compute $f(x)$ in polynomial time given x and $g(x)$, it suffices to show that $g \in \Sigma_k^{MM}$.

Suppose M runs in time $p(n)$ for some polynomial p. Let $w \in \Sigma^*$ and let $m = p(|w|)$. Then, let $x = x_1 \cdots x_m \in \{0,1\}^m$ be a sequence of possible oracle answers for $M(w)$. The question, "given w and x, run $M(w)$ using $x_1 \cdots x_m$ for the oracle answers, compute all of the queries, and ask are all of the 'yes' answers correct?" is a single Σ_k^p question. Thus there is a polynomial time computable predicate R such that "all of the 'yes' answers in $x_1 \cdots x_m$ are correct" if and only if $\exists |y_1| \leq m \; \forall |y_2| \leq m \cdots Q|y_k| \leq m \; R(w, x, y_1, \ldots, y_m)$.

By a similar argument to theorem 3.1, the maximum x such that all of the 'yes' answers in x are correct is the sequence of correct oracle answers

for $M(w)$. Thus, we can compute $M(w)$'s correct oracle answers by a Σ_k^{MM} machine in the following manner. Take the maximum over all $|x| = m$ of the maximum of all $|y_1| \leq m$ of the minimum of all $|y_2| \leq m$ of $\cdots$ of the maximum (or minimum) of all $|y_k| \leq m$ of either x if $R(w, x, y_1, \cdots, y_k)$ is true or else 0. For a fixed y_1, ..., y_{k-1}, the minimum over the y_k's is either x if $\forall y_k\, R(w, x, y_1, \ldots, y_k)$ or otherwise 0. Continuing this argument backwards for y_{k-1}, y_{k-2}, ..., y_1, we see that this strategy computes $M(w)$'s correct oracle answers, and hence $g \in \Sigma_k^{MM}$. □

5.3 Completeness Results

We now show that Σ_k^{MM} is an interesting and natural definition by giving natural complete problems for Σ_2^{MM} and Δ_3^p. First we show that MAXIMUM UNIVERSAL CIRCUIT is Σ_2^{MM} complete by a generic reduction, and then we reduce it to MAXIMUM DOUBLE KNAPSACK. The essential idea for MAXIMUM DOUBLE KNAPSACK came from the Σ_2^p completeness proof of INEQUIVALENCE OF CFG'S OVER A SINGLE LETTER ALPHABET [Hu82]. Converting the INEQUIVALENCE problem to an optimization problem yielded MAXIMUM DOUBLE KNAPSACK. Also, the idea for the Δ_3^p completeness result for DOUBLE KNAPSACK came from the analogy that an OptP completeness result gives rise to a Δ_2^p completeness result. Thus it is natural

to expect a close connection between a Σ_2^{MM} completeness result and a Δ_3^p completeness result.

Theorem 5.2 *The following problems are* Σ_2^{MM} *complete.*

- MAXIMUM UNIVERSAL CIRCUIT

  instance: Boolean circuit $C(x_1, \ldots, x_n, y_1, \ldots, y_n)$.

  output: Max $x_1 \cdots x_n$ such that $\forall y_1 \cdots \forall y_n \; C(x_1, \ldots, x_n, y_1, \ldots, y_n)$.

- MAXIMUM DOUBLE KNAPSACK

  instance: Positive integers $x_1, \ldots, x_n, y_1, \ldots, y_m, N$.

  output: The largest $M \leq N$ such that $M = \sum_{i \in S_1} x_i$ for some $S_1 \subseteq \{1, \ldots, n\}$ and $M \neq \sum_{j \in S_2} y_j$ for all $S_2 \subseteq \{1, \ldots, m\}$.

Proof: MAXIMUM UNIVERSAL CIRCUIT. The problem is clearly in Σ_2^{MM}; we show hardness by a generic reduction. Given M, a Σ_2^{MM} machine, and input $|w| = n$, construct circuit C as follows. Inputs $X = x_1, \ldots, x_m$ represent M's choices for its maximum alternations and the value M outputs on a given branch, and $Y = y_1, \ldots, y_m$ represent M's choices for its minimum alternations. Then, circuit $C(X, Y)$ says that X and Y encode a legal computation and that the value M outputs is in X. By encoding $M(w)$'s value in the high-order bits of X, we see that the value of $M(w)$ can be recovered from the maximum X such that $\forall Y \; C(X, Y) = 1$. Hence, the MAX UNIV CIRCUIT problem is Σ_2^{MM} complete. □

Proof: MAXIMUM DOUBLE KNAPSACK. We first show that the problem is in Σ_2^{MM}. Given an instance of MAX DOUBLE KNAPSACK, we can express M as the maximum over all $S_1 \subseteq \{1,\ldots,n\}$ of the minimum over all $S_2 \subseteq \{1,\ldots,m\}$ of either $\sum_{i\in S_1} x_i$ if $\sum_{j\in S_2} y_j \neq \sum_{i\in S_1} x_i \leq N$ or else 0. Then, for a given S_1, the minimum over all S_2 is either $\sum_{i\in S_1} x_i$ if $\sum_{i\in S_1} x_i \leq N$ and $\sum_{i\in S_1} x_i \neq \sum_{j\in S_2} y_j$ for all S_2 or otherwise 0. Thus the maximum over all S_1 of the minimum over all S_2 is the value of the MAX DOUBLE KNAPSACK problem.

We show MAX DOUBLE KNAPSACK is hard for Σ_2^{MM} by a metric reduction from MAXIMUM UNIVERSAL CIRCUIT. The first step is to modify MAX UNIV CIRCUIT slightly. In order to simplify the notation, we will write X and Y for $x_1,\ldots,x_n$ and $y_1,\ldots,y_n$, etc. Given a circuit C with inputs x_1, ..., x_n, y_1, ..., y_n, we rewrite $\varphi(X) = \forall Y\,(C(X,Y) = 1)$ as $\forall Y\,\neg(C(X,Y) = 0)$.

Now, the predicate $C(X,Y) = 0$ is certainly decidable in polynomial time, so by Cook's theorem, there is a 3CNF formula ψ such that $\varphi(X) = \forall Y\,\neg(\exists W \psi(X,Y,W) = 1) = \forall Y \forall W\,\neg(\psi(X,Y,W) = 1)$. Thus, without loss of generality, we may assume that C has the form $\neg\psi$, where ψ is in 3CNF.

The second step is to reduce τ to K, an instance of the KNAPSACK PROBLEM. We use the same reduction as in the OptP completeness proof

Table 5.1: DOUBLE KNAPSACK construction.

	x_1^1 x_i^1 x_n^1	y_1 y_i y_n	C_1^1 C_j^1 C_m^1	C_1^2 C_j^2 C_m^2	x_1^2 x_i^2 x_n^2
x_i	1		1 for each clause x_i is in		1
$\overline{x}_i$	1		1 for each clause $\overline{x}_i$ is in		0
y_i		1	1 for each clause y_i is in		
$\overline{y}_i$		1	1 for each clause $\overline{y}_i$ is in		
C_j^0			0	1	
C_j^1			1	1	
C_j^2			2	1	
x_i'					1
N	1 ··· 1	1 ··· 1	3 ··· 3	1 ··· 1	1 ··· 1

for KNAPSACK with the following addition. Also include, for each variable x_i, an integer x_i' with a 1 in the x_i^2 zone. The purpose of the x_i''s is to use up the slack between M and N. The entire construction is summarized in table 5.1.

Thus a solution to K that sums exactly to N corresponds to a satisfying assignment for τ. Conversely, given a satisfying assignment to τ, we can find a solution to K by choosing the integers corresponding to the true

literals in τ. Thus, K has a solution if and only if τ is satisfiable. This also implies that a given assignment to $x_1 \cdots x_n$ can be extended to a satisfying assignment in τ if and only if there is a subset of the y_i, $\overline{y}_i$, C_i^0, C_i^1, C_i^2, x_i' integers that sums to N minus the sum associated with the assignment to $x_1 \cdots x_n$. And $x_1 \cdots x_n$ can *not* be extended to a satisfying assignment if and only if $\forall Y \neg\tau(X, Y)$. Thus, we have reduced the problem to: given integers $a_1, \ldots, a_n, b_1, \ldots, b_m, N$, find the largest sum $\sum_{i \in S_1} a_i$ such that no subset of the b_j's satisfies

$$\sum_{i \in S_1} a_i + \sum_{j \in S_2} b_j = N.$$

The final step in the construction is to reduce this last problem to MAX DOUBLE KNAPSACK. First, notice in the above construction, it turns out that the b_j's are all much smaller than the a_i's. In fact, we may assume that $b_j \leq N/m$. Replace the b_j's by $z_1 = N/m - b_1, \ldots, z_m = N/m - b_m$, $z_{m+1} = \cdots = z_{2m} = N/m$. Then, any integer larger than $N - N/m$ can be expressed as $N - \sum b_j$ if and only if it can be expressed as $\sum z_j$. Thus we have reduced this last problem to MAX DOUBLE KNAPSACK. □

Theorem 5.3 *The following problem is Δ_3^p complete.*

DOUBLE KNAPSACK

instance: Positive integers $x_1, \ldots, x_n, y_1, \ldots, y_m, N, k$.

question: Is the k^{th} bit of M a 1, where M is defined as in the MAXIMUM DOUBLE KNAPSACK problem?

Proof: By a generic reduction from an arbitrary Δ_3^p language. Let $L \in \Delta_3^p$ and let M be a $\mathsf{P}^{\Sigma_2^p}$ machine recognizing L. By theorem 5.1, there is a function $f \in \Sigma_2^{MM}$ such that $f(x)$ is the true oracle answers in $M(x)$'s computation. A simple modification to this theorem lets us also assume that the lowest order bit of $f(x)$ is a 1 if $M(x)$ accepts and a 0 if $M(x)$ rejects. Thus L reduces to the problem of computing the least significant bit of f.

Looking at the Σ_2^{MM}-completeness proof of the MAX UNIV CIRCUIT problem, we see that in the reduction the value of f is represented in the high-order bits of the MAX UNIV CIRCUIT problem. Thus the problem of computing a bit of f reduces to the problem of computing a bit of MAX UNIV CIRCUIT.

The final step is to reduce this last problem to DOUBLE KNAPSACK. Again, looking at the reduction, we see that the problem of computing a bit of MAX UNIV CIRCUIT reduces to the problem of computing a bit of MAX DOUBLE KNAPSACK. Thus L reduces to DOUBLE KNAPSACK and hence DOUBLE KNAPSACK is Δ_3^p complete. □

Chapter 6

Discussion

We conclude with a discussion of possible directions for future work. The main point of this thesis is that NP-complete problems possess a deeper level of structure than was previously known. It is natural to ask if there is even a deeper level of structure in these problems. We have considered problems at the level of computing the value of the optimal solution; surely, the level of their feasible solutions possesses an equally rich structure. It is just possible that a good understanding of the structure of the feasible solutions in NP-complete problems might yield some insight into how well they can or cannot be approximated.

In fact, I got into this area by looking at approximation algorithms and trying to understand why some problems could be approximated better than others. I started by digging into the reductions among these problems

to see if they were any help. This quickly led to the realization that neither the ratio of the costs of approximate solutions nor their additive difference was the key measure that was being preserved between problems. I was trying to separate an inner core of structure from the more arbitrary cost function assigned to the feasible solutions. I regard this thesis as a partial answer to that goal. This eventually led to the idea that it was the number of NP queries embedded in the optimal answer that was being preserved across reductions. And so when it turned out that not all problems were the same in this new measure — it was fantastic!

Another idea is to explore the connection between the classes NP, D^p, Δ_2^p, #P and OptP. We showed that under very general conditions, an OptP-completeness result yielded completeness results for NP, D^p and Δ_2^p and that this approach could be used for many natural problems. Are there other results that tie these classes together? For example, can problems be put in a framework where one version is OptP-complete if and only if another version is #P-complete?

It would also be quite interesting to find natural complete problems for subclasses of $\mathsf{FP}^{\mathrm{SAT}}$ other than $\mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ and $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$. With the peculiar exception of BIN PACKING, almost all other natural problems seem to fall in one of these two categories. Exhibiting a natural complete problem for a different subclass would firmly establish the importance of

$\mathsf{FP}^{\mathrm{SAT}}$ as a complexity measure. Gasarch [Ga86] poses a very good question along these lines. He notes that the complexity of actually *finding* the largest clique in a graph (not just giving its size) is not at all clear. The problem is in $\mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ and is hard for $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$, but its precise complexity within these bounds is not known.

A more technical problem is to determine the precise complexity of BIN PACKING. We know that BIN PACKING is in $\mathsf{FP}^{\mathrm{SAT}}[O(\log\log n)]$ and that it is hard for $\mathsf{FP}^{\mathrm{SAT}}[1]$, but no better bounds are known. Another technical problem is to extend the separation results in chapter 4 above $\log n$. Is it true that $\mathsf{P} \neq \mathsf{NP}$ implies that $\mathsf{FP}^{\mathrm{SAT}}[f(n)]$ is properly contained in $\mathsf{FP}^{\mathrm{SAT}}[g(n)]$ whenever $f(n) < g(n)$? It is not even known if $\mathsf{FP}^{\mathrm{SAT}}[\log^2 n] = \mathsf{FP}^{\mathrm{SAT}}[n]$ implies $\mathsf{P} = \mathsf{NP}$. Alternatively, since the separation theorems relativize, are there oracles where these classes are equal but $\mathsf{P} \neq \mathsf{NP}$?

Lastly, another idea is to find other associative operators besides the MAX, PLUS, AND and OR functions that define interesting classes of problems when applied to the branches of an NP machine. Considering the EXCLUSIVE OR operator, for example, PARITY SAT, the set of boolean formulas with an even number of satisfying assignments, is a natural complete problem. Valiant and Vazirani [VV85] show that PARITY SAT is hard for both NP and coNP under randomized reductions, and they ask where in the polynomial-time hierarchy it lies.

Bibliography

[AHU74] Aho, A., J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[BGS75] Baker, T., J. Gill and R. Solovay, "Relativizations of the P =? NP Question," *SIAM J. of Computing* **4**, pp. 431–442, 1975.

[Be86] Beigel, R., "Query-Limited Reducibilities," Ph.D. Thesis, Dept. of Computer Science, Stanford University, 1986.

[BH77] Berman, L. and J. Hartmanis, "On Isomorphisms and Density of NP and Other Complete Sets," *SIAM J. of Computing* **6**, pp. 305–327, 1977.

[CKS81] Chandra, A., D. Kozen and L. Stockmeyer, "Alternation," *JACM* **28**, pp. 114–133, 1981.

[Ch76] Christophides, C., "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem," Tech. Report, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[Co71] Cook, S., "The Complexity of Theorem-Proving Procedures," *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pp. 151–158, 1971.

[GJ76] Garey, M. and D. Johnson, "The Complexity of Near-Optimal Graph Coloring," *JACM* **23**, pp. 43–49, 1976.

[GJ79] Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.

[GJS76] Garey, M., D. Johnson, and L. Stockmeyer, "Some Simplified NP Complete Graph Problems," *Theoretical Computer Science* **8**, pp. 237–267, 1976.

[Ga86] Gasarch, W., "The Complexity of Optimization Functions," Tech. Report 1652, Dept. of Computer Science, University of Maryland, 1986.

[Hu82] Huynh, T.-D., "Deciding the Inequivalence of Context-Free Grammars with One-Letter Terminal Alphabet is Σ_2^p-Complete," *Proc. 23rd Ann. IEEE Symp. of Foundations of Computer Science*, pp. 21–31, 1982.

[Jo85] Johnson, D., "The NP-completeness Column: An Ongoing Guide," *J. of Algorithms*, June 1985.

[Ka86] Kadin, J., "Deterministic Polynomial Time with O(log n) Queries," Tech. Report 86-771, Dept. of Computer Science, Cornell University, 1986.

[KK82] Karmarkar, N. and R. Karp, "An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem," *Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science*, pp. 312–320, 1982.

[Ka72] Karp, R., "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, eds., Plenum Press, New York, pp. 85–103, 1972.

[Pa84] Papadimitriou, C., "On the Complexity of Unique Solutions," *JACM* **31**, pp. 392–400, 1984.

[PY84] Papadimitriou, C. and M. Yannakakis, "The Complexity of Facets (and Some Facets of Complexity)," *JCSS* **28**, pp. 244–259, 1984.

[Sk85] Skiena, S., "Complexity of Optimization Problems on Solitaire Game Turing Machines," M.S. Thesis, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaigne, 1985.

[St77] Stockmeyer, L., "The Polynomial-Time Hierarchy," *Theoretical Computer Science* **3**, pp. 1–22, 1977.

[Va79] Valiant, L., "The Complexity of Computing the Permanent," *Theoretical Computer Science* **8**, pp. 189–201, 1979.

[VV85] Valiant, L. and V. Vazirani, "NP is as Easy as Detecting Unique Solutions," *Proc. 17th Ann. ACM Symp. of Theory of Computing*, 1985.