JOURNAL OF COMPUTER AND SYSTEM SCIENCES 36, 490–509 (1988)

The Complexity of Optimization Problems

MARK W. KRENTEL*,†

Computer Science, Cornell University, Ithaca, New York 14853

Received August 5, 1986; revised September 9, 1987

We consider **NP**-complete optimization problems at the level of computing their optimal value, and define a class of functions called **OptP** to capture this level of structure. We show that TRAVELING SALESPERSON and KNAPSACK are complete for **OptP**, and that CLIQUE and COLORING are complete for a subclass of **OptP**. These results show a deeper level of structure in these problems than was previously known. We also show that **OptP** is closely related to $\mathbf{FP}^{\mathrm{SAT}}$, the class of functions computable in polynomial time with an oracle for **NP**. This allows us to quantify exactly "how much" **NP**-completeness is in these problems. In particular, in this measure, we show that TRAVELING SALESPERSON is strictly harder than CLIQUE and that CLIQUE is strictly harder than BIN PACKING. A further result is that an **OptP**-completeness result implies **NP**-, $\mathbf{D}^p$-, and Δ_2^p-completeness results, thus tying these four classes closely together.

1. INTRODUCTION

We consider **NP**-complete optimization problems, such as CLIQUE, COLORING and TRAVELING SALESPERSON, at the level of computing their optimal value, and we define a class of functions to capture this level of structure. The traditional approach to these problems has been to study the complexity of an equivalent yes/no question. For example, the TRAVELING 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 and an integer k, does G have a TSP tour of cost at most k? And although this transformation loses some of the structure of the original problem, we say that it captures the essential difficulty of the TSP problem because we can solve the original problem by using the yes/no question as a subroutine.

In this paper, we study **NP**-complete problems and focus on the deeper level of structure of actually computing the optimal value. In order to capture this level of structure, 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

* This work was partially funded by NSF Grant BCR 85-03611.

† Current address: Rice University, PO Box 1892, Houston TX 77251.

0022-0000/88 $3.00

restricted to $z(n)$ bits of output. For example, the size of the largest clique in a graph can be computed in this manner. Consider any subset 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 [22] used an analogous approach in defining the class of functions $\#\,\mathbf{P}$ 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 Section 2 we show that the problems of computing the optimal value for TRAVELING 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 strengthen the **NP**-completeness proofs for these problems and show that one problem can simulate the optimal value of another problem, more than just its yes/no question. Skiena [20] defines a Solitaire game Turing machine and the complexity classes **SGP** and **SG'P**, identical to our notions of **OptP** and $\mathbf{OptP}[O(\log n)]$. He also independently proves several problems complete for **SGP** and **SG'P**, including many of the problems we give in Section 2.

In Section 3 we define $\mathbf{FP}^{\mathrm{SAT}}[z(n)]$, the class of functions computable in polynomial time with $z(n)$ queries to an **NP** oracle. It is straightforward to show that $\mathbf{FP}^{\mathrm{SAT}}[z(n)]$ contains $\mathbf{OptP}[z(n)]$ 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 $f \in \mathbf{FP}^{\mathrm{SAT}}[z(n)]$ can be reduced to some $g \in \mathbf{OptP}[z(n)]$. 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 [10] also considers the number of queries it takes to compute various functions and defines the class $Q[z(n), \mathbf{NPC}]$, identical to our notion of $\mathbf{FP}^{\mathrm{SAT}}[z(n)]$. He independently proves hardness results for many **NP** functions, but unfortunately, his lower bound techniques do not allow for successive queries to depend on the answers to the previous ones and he ends up with somewhat weaker bounds.

This result also allows us to tie together the classes **NP**, $\mathbf{D}^p$, and Δ_2^p. We show that under very general conditions, an **OptP**-completeness result also yields **NP**-, $\mathbf{D}^p$-, 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 TRAVELING SALESPERSON problem, the question "Is the optimal value at most k?" is **NP**-complete [15], the question "Is the optimal value equal to k?" is $\mathbf{D}^p$-complete [19], and the question "Is the optimal solution unique?" is Δ_2^p-complete [18]. In Section 3 we show that these results for **NP** and $\mathbf{D}^p$ and a similar result for Δ_2^p can be obtained as corollaries of the single **OptP**-completeness result for TSP.

In Section 4 we consider separation results among $\mathbf{FP}^{\mathrm{SAT}}$ classes. Assuming $\mathbf{P} \neq \mathbf{NP}$, we show that $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$ is strictly contained in $\mathbf{FP}^{\mathrm{SAT}}[O(n)]$ and

also that $\mathbf{FP}^{\mathrm{SAT}}[f(n)]$ is strictly contained in $\mathbf{FP}^{\mathrm{SAT}}[g(n)]$ for "sufficiently nice" f and g whenever $f(n) < g(n)$ and $f(n) \leqslant \frac{1}{2}\log n$. Since TSP is complete for $\mathbf{FP}^{\mathrm{SAT}}[O(n)]$ and since CLIQUE is complete for $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$, this result shows that TSP is strictly harder than CLIQUE in this measure. Karmarkar and Karp [14] 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 $\mathbf{FP}^{\mathrm{SAT}}[O(\log\log n)]$ 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.

It appears that our measure of the relative complexities of **NP**-complete problems, namely the number of **NP** queries needed to determine the optimal value, does not directly correspond to other methods. For example, KNAPSACK is complete for $n^{O(1)}$ queries but is solvable in pseudo-polynomial time and hence is not strongly **NP**-complete [8]. On the other hand, BIN PACKING *is* strongly **NP**-complete but can be solved with $O(\log\log n)$ queries by using the approximation algorithm of Karmarkar and Karp [14]. Our classification also does not seem to correspond to approximation algorithms and worst-case performance ratios. For example, although TRAVELING SALESPERSON and KNAPSACK are both **OptP**-complete, TSP (without the triangle inequality) cannot be approximated to within any constant factor, while KNAPSACK has a fully polynomial-time approximation scheme.

2. Optimization Problems

2.1. *Introduction*

The goal of this section is to consider **NP**-complete optimization problems such as TRAVELING SALESPERSON, CLIQUE, and COLORING and show that they possess a deeper level of structure. The problems we consider here were, of course, already known to be **NP**-complete. The **NP**-completeness results show that these problems are all equivalent at the level of their yes/no questions. Here, we are considering problems at the level of computing their optimal value, and we will see that there is more than one equivalence class. We start with some preliminary notation, and then we define metric Turing machines and the class **OptP** in order to capture our intuitive notion of an optimization problem.

Notation. We write $\mathbf{N} = \{0, 1, 2, \ldots\}$ for the set of natural numbers and $\mathbf{Q}$ for the set of rationals. Σ denotes a fixed finite alphabet; without loss of generality, we can take $\Sigma = \{0, 1\}$. We also take $\log n$ to mean the base 2 logarithm of n.

Definition. An **NP** *metric Turing Machine*, N, is a nondeterministic 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 $\mathbf{OptP}[z(n)]$ if $f \in \mathbf{OptP}$ 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 $\mathbf{OptP}[n^{O(1)}]$. 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**) [22]. 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** functions 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 $x \in \Sigma^*$.

DEFINITION. A function f is *hard* for a complexity class $\mathscr{C}$ if all $g \in \mathscr{C}$ are reducible to f; and f is *complete* for $\mathscr{C}$ if f is hard for $\mathscr{C}$ and also $f \in \mathscr{C}$.

Note that we need a many–one reduction here. Eventually (see Section 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 $\mathbf{OptP}[z(n)]$, then f is also complete for $\mathbf{OptP}[z(n^{O(1)})]$. And lastly, note that metric reductions are closed under composition.

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

DEFINITION. A function $z: \mathbf{N} \to \mathbf{N}$ is *smooth* if the function $1^n \mapsto 1^{z(n)}$ is computable in polynomial time and if $z(x) \leqslant z(y)$ for all $x \leqslant y$.

LEMMA. *Let $z(n)$ be smooth. The following functions are complete for* **OptP**$[z(n)]$ *under metric reductions*:

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

instance. *$N \# x \# 0^k$, where N is a description of an* **NP** *metric Turing machine.*
output. $\text{UNIV}_{z(n)}$ *simulates the branches of $N(x)$ and simulates each branch 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 $\varphi(x_1, ..., x_n)$.*
output. *The lexicographically maximum $x_1 \cdots x_m \in \{0, 1\}^m$ that can be extended to a satisfying assignment where $m = \min\{n, z(|\varphi|)\}$.*

Proof. UNIV. First, UNIV_z is clearly in **OptP**$[z(n)]$ because it is defined as the output of an **NP** metric Turing machine that outputs at most $z(n)$ bits. We show hardness by a generic reduction. Let $f \in$ **OptP**$[z(n)]$, let N be an **NP** metric Turing machine 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. Again, LEX_z is clearly in **OptP**$[z(n)]$. Since metric reductions are closed under composition, to show hardness, it suffices to reduce UNIV_z to LEX_z. Let $y = N \# w \# 0^k$ be an instance of UNIV_z. By Cook's theorem [7], there is a 3CNF boolean formula $\varphi(x_1, ..., x_n)$, with $|\varphi|$ polynomial in $|y|$, which says that "$x_1 \cdots x_n$ encode a valid computation of some branch of $\text{UNIV}_z(y)$ and $x_1 \cdots x_{z(|y|)}$ represent the output on this branch." Then, $\mathbf{opt}^{\text{UNIV}}(y)$ can be obtained from the first $z(|y|)$ bits of $\mathbf{opt}^{\text{LEX}}(\varphi)$ (certainly, $z(|y|) \leqslant z(|\varphi|)$), 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, TRAVELING 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 (W. SAT):

instances. *Boolean formula in conjunctive normal form* (CNF) *with (binary) weights on the clauses.*

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

- TRAVELING SALESPERSON (TSP):

instance. *Graph G with integer weights on the edges.*
output. *The length of the shortest traveling salesperson tour in G.*

- MAXIMUM SATISFYING ASSIGNMENT (MSA):

instance. *Boolean formula* $\varphi(x_1, ..., x_n)$.
output. *The lexicographically maximum* $x_1 \cdots x_n \in \{0, 1\}^n$ *that satisfies* φ, *or* 0 *if* φ *is not satisfiable.*

- 0–1 INTEGER LINEAR PROGRAMMING (01ILP):

instance. *Integer matrix A and integer vectors B and C.*
output. *The maximum value of* $C^T X$ *over all* 0–1 *vectors X subject to* $AX \leqslant B$.

- KNAPSACK:

instance. *Integers* $x_1, ..., x_n$, N.
output. *The largest value, less than N, of* $\sum_{i \in S} x_i$ *taken over all* $S \subseteq \{1, ..., n\}$.

Proof. WEIGHTED SATISFIABILITY. All of the problems in this section are easily seen to be in **OptP**, and all of the reductions in this section can be computed in polynomial time. The previous lemma shows that UNIV_n is **OptP**$[n]$-complete; it is also **OptP**$[n^{O(1)}]$-complete by sufficiently padding the input to an arbitrary **OptP** function. Thus it suffices to reduce UNIV_n to WEIGHTED SATISFIABILITY.

Let $x \in \Sigma^*$ be an instance of UNIV_n, let $n = |x|$ and define the boolean formula $\varphi_x(z_1, ..., z_m, y_1, ..., y_n)$ to mean "$z_1, ..., z_m$ encode a valid 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 [7], we can encode φ_x as a CNF formula where m is polynomial in n, the length of x. The presentation of Cook's theorem in [1] shows how to encode a Turing machine computation with a CNF boolean formula.

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. Also, because the weight on each clause in φ_x is greater than the sum of the weights on the y_i's, 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,

$$\mathbf{opt}^{\text{UNIV}}(x) = \mathbf{opt}^{\text{W. SAT}}(\Phi_x) - k2^{2n},$$

where k = the number of clauses in φ_x. This gives a metric reduction from UNIV_n to WEIGHTED SATISFIABILITY, and hence WEIGHTED SATISFIABILITY is complete for **OptP**. ∎

Proof. TRAVELING SALESPERSON. We reduce WEIGHTED SATISFIABILITY to TSP. The reduction is in two parts: first we reduce W. SAT to CONSTRAINED TSP and then we remove the constraints. Papadimitriou used the same technique to show that the

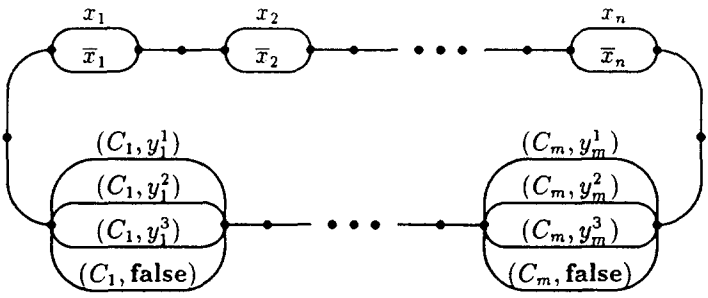

FIG. 1. TSP construction.

problem of deciding if an instance of TSP has a unique optimum solution is Δ_2^p-complete [18].

Suppose φ is an instance of W. SAT with variables $x_1, ..., x_n$, and clauses $C_1, ..., C_m$ with weights $w_1, ..., w_m$. 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 each variable x_i, include a pair of nodes and two edges between them: one edge labeled x_i and one labeled $\bar{x}_i$. It will turn out that any tour in G will use either the x_i or the $\bar{x}_i$ 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 (we claim without proof that W. SAT remains **OptP**-complete for formulas in 3CNF, thus we may assume that C_j has at most 3 literals), 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 labeled $(C_j, \textbf{false})$. 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 Fig. 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 nonedges 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 $\bar{x}_i$ in clause C_j, include a NAND constraint between x_i and $(C_j, \bar{x}_i)$. Also include a NAND constraint for each pair $\bar{x}_i$ 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 cannot happen in an optimal tour.) Thus,

$$\mathbf{opt}^{\text{W. SAT}}(\varphi) = \sum_{i=1}^{m} w_i - \mathbf{opt}^{\text{TSP}}(G).$$

This gives a metric reduction from WEIGHTED SATISFIABILITY 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 [18]. ∎

Proof. MAXIMUM SATISFYING ASSIGNMENT. MSA is the same problem as LEX_n. ∎

Proof. 0–1 INTEGER LINEAR PROGRAMMING. We reduce W. SAT to 01ILP. Let φ be a weighted formula with variables $x_1, ..., x_n$, clauses $C_1, ..., C_m$ and weight $w_1, ..., w_m$. Again, we may assume that φ is in 3CNF. Reduce φ to an instance I of 01ILP with variables $x_1, ..., x_n, \bar{x}_1, ..., \bar{x}_n$, and $c_1, ..., c_m$. For each variable x_i, include the constraint $x_i + \bar{x}_i = 1$, 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 [8]. 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 digits 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 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 $\bar{x}_i$; 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 I. 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

TABLE I

KNAPSACK construction. Entries not explicitly stated are 0.

	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	
$\bar{x}_i$		1		1 for each clause $\bar{x}_i$ is in							0	
C_j^0					0			1				
C_j^1					1			1				
C_j^2					2			1				
N	1	⋯	1	3	⋯	3	1	⋯	1	1	⋯	1
M	1	⋯	1	3	⋯	3	1	⋯	1	0	⋯	0

weight $x_i = \textbf{true}$ heavier than $x_i = \textbf{false}$; it will turn out that 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 assignment 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 do not count), and since there are $2n+3m$ numbers, setting $r = 2(2n+3m)+1$ will guarantee that there are no possibilities of any carries (see Table I).

Thus we see that a solution to K that sums to a value at least M and at most N 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. We may assume that the all-**false** assignment satisfies φ. Thus the optimal value for K is greater than M if and only if φ has a satisfying assignment. Furthermore, by weighting $x_i = \textbf{true}$ heavier than $x_i = \textbf{false}$ in the x_i^2 zone, we see that

$$\textbf{opt}^{\text{MSA}}(\varphi) = \textbf{opt}^{\text{KNAPSACK}}(K) - M.$$

Thus, KNAPSACK is **OptP**-complete. ∎

2.3. **OptP**[$O(\log n)$]-*Completeness Results*

In this section, we give the reductions to show that MAXIMUM SATISFIABILITY, 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 [15] 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 [9] constructs a graph with chromatic 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 (MAX SAT):

instance. CNF *boolean formula.*
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. Again, all of the problems in this section are clearly in $\mathbf{OptP}[O(\log n)]$, and all of the reductions can be computed in polynomial time. Also, $\text{UNIV}_{\log n}$ is complete for $\mathbf{OptP}[O(\log n)]$ so it suffices to reduce $\text{UNIV}_{\log n}$ to MAX SAT.

The proof for W. SAT can be modified to give a reduction from $\text{UNIV}_{\log n}$ to MAX SAT. Since the output is only $\log n$ bits long, the weights on the clauses need only be in the range $1 \cdots O(n)$. Then, since the weights are only polynomially large, they can be removed by repeating clauses, and the reduction will still produce a formula that is only polynomially long. ∎

Proof. CLIQUE. The reduction from SATISFIABILITY to CLIQUE given in [15] or in [1] 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 construct a graph G_n from a boolean formula φ such that $\chi(G_n) = 3n$ if $\varphi \in \text{SAT}$ and $\chi(G_n) = 4n$ if $\varphi \notin \text{SAT}$ (we write $\chi(G)$ for the *chromatic number* of G). Given a boolean formula φ, [9] shows how to construct a graph G such that $\chi(G) = 3$ if $\varphi \in \text{SAT}$ and $\chi(G) = 4$ if $\varphi \notin \text{SAT}$. Let G' be the composition graph, $K_n[G]$, consisting of n copies of G, where vertices within the same copy have the same edge structure as in G and all pairs of vertices in different copies are connected by an edge. It is straightforward to verify that $\chi(G') = 3n$ if $\varphi \in \text{SAT}$ and $\chi(G') = 4n$ if $\varphi \notin \text{SAT}$.

Now let Φ be a CNF formula with n clauses. Again from Cook's theorem, we can construct boolean formulas $\varphi_2, \ldots, \varphi_{n+1}$ such that φ_i is satisfiable if and only if Φ has at least i simultaneously satisfiable clauses. (Assume without loss of generality, that Φ has at least 2 simultaneously satisfiable clauses.) By the previous paragraph, we can construct graphs $G_2, \ldots, G_{n+1}$ such that

$$\chi(G_i) = \begin{cases} 12n - 3i & \text{if } \varphi_i \text{ is satisfiable} \\ 16n - 4i & \text{if } \varphi_i \text{ is not satisfiable.} \end{cases}$$

Define $G^* = G_2 + \cdots + G_{n+1}$, the disjoint union of the G_i's, so that $\chi(G^*) = \max_i \chi(G_i)$, and let $k = \mathbf{opt}^{\text{MAX SAT}}(\Phi)$. Then, $\varphi_2, \ldots, \varphi_k$ are satisfiable but $\varphi_{k+1}, \ldots, \varphi_{n+1}$ are not, so $\chi(G^*) = \chi(G_{k+1}) = 16n - 4k - 4$ and hence

$$\mathbf{opt}^{\text{MAX SAT}}(\Phi) = 4n - 1 - \tfrac{1}{4}\mathbf{opt}^{\text{COLORING}}(G^*).$$

All of these graphs can be constructed in polynomial time, so COLORING is $\mathbf{OptP}[O(\log n)]$-complete. ∎

Proof. LONGEST CYCLE. The reduction from WEIGHTED SATISFIABILITY to TRAVELING SALESPERSON can be modified to give a reduction from MAX SAT to LONGEST CYCLE. First, change the TSP problem so that we are looking for the *longest* tour. Then, construct the same constrained graph and notice that the weights on

the edges are only polynomially large. Remove the weights by repeating edges and then remove the constraints. This will achieve

$$\mathbf{opt}^{\text{MAX SAT}}(\varphi) = \mathbf{opt}^{\text{LONGEST CYCLE}}(G) - \text{const}_{\varphi}$$

and hence LONGEST CYCLE is $\mathbf{OptP}[O(\log n)]$-complete. ∎

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

3.1. *Introduction*

In this section, we consider functions computable in polynomial time with an oracle for **NP**, and we show that they are closely related to **OptP** functions. 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.

DEFINITION. A function $f: \Sigma^* \to \mathbf{N}$ is in $\mathbf{FP}^{\text{SAT}}$ if f is computable in polynomial time with an oracle for **NP**. We say that f is in $\mathbf{FP}^{\text{SAT}}[z(n)]$ if $f \in \mathbf{FP}^{\text{SAT}}$ and f is computable using at most $z(n)$ queries on inputs of length n.

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

Note again that $\mathbf{FP}^{\text{SAT}} = \mathbf{FP}^{\text{SAT}}[n^{O(1)}]$ and that $\mathbf{P}^{\text{SAT}} = \mathbf{P}^{\text{SAT}}[n^{O(1)}]$. Our main result is that an $\mathbf{FP}^{\text{SAT}}$ function metrically reduces to an **OptP** function. This result says that an $\mathbf{OptP}[f(n)]$-complete function is also complete for $\mathbf{FP}^{\text{SAT}}[f(n)]$, and it sheds new light on the structure discussed in Section 2. Thus, TRAVELING SALESPERSON, WEIGHTED SATISFIABILITY, MAXIMUM SATISFYING ASSIGNMENT, 0–1 INTEGER LINEAR PROGRAMMING and KNAPSACK are all complete for $\mathbf{FP}^{\text{SAT}}[n^{O(1)}]$, and MAXIMUM SATISFIABILITY, CLIQUE, COLORING, and LONGEST CYCLE are all complete for $\mathbf{FP}^{\text{SAT}}[O(\log n)]$. This gives a precise characterization of "how much" **NP**-completeness these problems contain.

It would have been possible to prove these last results directly, without going through Section 2. The reason for organizing them as we have is that this is how optimization problems naturally arise, and that the results in Section 2 could stand on their own.

3.2. *Main Result*

First we show that every function in $\mathbf{FP}^{\text{SAT}}$ 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 $\mathbf{P}^{\text{SAT}}$ machine. An **NP** machine could guess the entire $\mathbf{P}^{\text{SAT}}$ computation and could even

verify the "yes" answers, but it has no may 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 correct oracle answers in the $\mathbf{P}^{\mathrm{SAT}}$ computation.

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

Proof. Let $f \in \mathbf{FP}^{\mathrm{SAT}}[z(n)]$, and let M compute f, where M is a $\mathbf{P}^{\mathrm{SAT}}$ 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 $b_1, ..., b_{z(n)} \in \{0, 1\}$, 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, $\varphi_1, ..., \varphi_{z(n)}$. 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 $\varphi_1 \in \mathrm{SAT}$, then $N(x)$ on branch $10 \cdots 0$ will find a satisfying assignment; so, $\mathbf{opt}^N(x) \geqslant 10 \cdots 0$ and thus $b_1 = 1$. On the other hand, if $\varphi_1 \notin \mathrm{SAT}$, 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, $\mathbf{opt}^N(x) \leqslant 01 \cdots 1$ 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. ∎

Because we can compute the value of an $\mathbf{OptP}[z(n)]$ function bit by bit with $z(n)$ queries, we see that the converse of Theorem 3.1 is immediate. Therefore we can offer (without proof) the following characterization of $\mathbf{P}^{\mathrm{SAT}}$ computations, for both functions and languages, in terms of $\mathbf{P}^{\mathrm{SAT}}$.

THEOREM 3.2. *Let z be smooth.*

(i) *$f \in \mathbf{FP}^{\mathrm{SAT}}[z(n)]$ if and only if f can be written as $f(x) = h(x, g(x))$ where $g \in \mathbf{OptP}[z(n)]$ and h is p-computable.*

(ii) *$f \in \mathbf{FP}^{\mathrm{SAT}}[z(n^{O(1)})]$ if and only if f is metrically reducible to some $g \in \mathbf{OptP}[z(n)]$.*

(iii) $L \in \mathbf{P}^{\mathrm{SAT}}[z(n)]$ *if and only if* L *can be written as* $L = \{x \mid P(x, g(x))\}$ *where* $g \in \mathbf{OptP}[z(n)]$ *and* P *is a p-computable predicate.*

Notice that Theorem 3.1 allows us to quantify "how much" **NP**-completeness is in a problem. Define the function QUERY that takes as input $x_1 \# \cdots \# x_n$ and outputs $b_1 \cdots b_n$, where $b_i = 1$ if $x_i \in \mathrm{SAT}$ and $b_i = 0$ if $x_i \notin \mathrm{SAT}$. Then, clearly $\mathrm{QUERY} \in \mathbf{FP}^{\mathrm{SAT}}$; so, by Theorem 3.1, QUERY is metrically reducible to TSP. This shows that the answers to n **NP** questions can be embedded in the optimal value of a single instance of TSP. A similar result can be obtained for CLIQUE by restricting the number of queries to $\log n$.

It is tempting to conjecture that QUERY is *complete* for $\mathbf{FP}^{\mathrm{SAT}}$, but this does not seem to be the case. The (subtle) problem with that idea is that the successive questions in QUERY do not depend on the answers to the previous questions. If we modified QUERY to allow for this, then we claim without proof that it *would* be complete for $\mathbf{FP}^{\mathrm{SAT}}$. Gasarch [10] shows that QUERY, as the problem is defined above, is hard for $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$.

3.3. *Applications*

A further result is that under very general conditions, **OptP**-complete problems give rise to complete problems for Δ_2^p, $\mathbf{D}^p$, and **NP**. This result allows us to tie these classes together in a stronger way than was previously known. We conjecture that if f is **OptP**-complete, then $\{x \# k \mid f(x) = k\}$ is $\mathbf{D}^p$-complete. Unfortunately, the proof does not 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 $T_1: \Sigma^* \to \Sigma^*$, $T_2: \Sigma^* \to \mathbf{Q} \setminus \{0\}$, 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 MAXIMUM SATISFYING ASSIGNMENT. (Although the reduction for KNAPSACK is from MAXIMUM SATISFYING ASSIGNMENT, the composition of reductions from UNIV_n 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 many-one complete for* Δ_2^p.

(ii) *If* f *is complete for* **OptP** *via linear reductions, then* $L_2 = \{x \# k_1 \# k_2 \mid f(x) \equiv k_1 \pmod{k_2}\}$ *is many-one 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 many-one complete for* $\mathbf{D}^P$.

(iv) *If f is hard for* **OptP**[1] *via linear reductions, then* $L_4 = \{x \# k \mid f(x) \geq k\}$ *is many-one 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 $g \in \mathbf{FP}^{\text{SAT}}$, 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 $L \in \mathbf{P}^{\text{SAT}}$ and let M be a $\mathbf{P}^{\text{SAT}}$ 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, $g \in$ **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 $g(x) \equiv 1 \pmod 2$?" reduces to a question of the form "Is $f(y) \equiv k_1 \pmod{k_2}$?"

(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, $g \in$ **OptP** [2]. Also, the $\mathbf{D}^p$ complete problem SAT-UNSAT [19] 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 $g \in$ **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 (without proof) natural complete languages for $\mathbf{P}^{\text{SAT}}(= \Delta_2^p)$ and $\mathbf{P}^{\text{SAT}}[O(\log n)]$. Previously, the only known example of a complete language for $\mathbf{P}^{\text{SAT}}$ was the UNIQUELY OPTIMAL TRAVELING SALESPERSON problem [18]. Kadin [13] discusses $\mathbf{P}^{\text{SAT}}[O(\log n)]$ and gives natural complete languages for this class.

THEOREM 3.4. *The following languages are many-one complete for* $\mathbf{P}^{\text{SAT}}$.

- **instance.** *Boolean formula* $\varphi(x_1, \ldots, x_n)$.
  question. *Is* $x_n = 1$ *in* φ*'s maximum satisfying assignment*?
- **instance.** *Weighted graph G and integer k.*
  question. *Is the length of the shortest traveling salesperson tour in G equivalent to* 0 mod k?

THEOREM 3.5. *The following languages are many-one complete for* $\mathbf{P}^{\text{SAT}}[O(\log n)]$.

- **instance.** *Boolean formula φ and integer k.*
  question. *Is the maximum number of simultaneously satisfiable clauses in φ equivalent to* $0 \bmod k$?
- **instance.** *Graph G and integer k.*
  question. *Is the size of the maximum clique in G equivalent to* $0 \bmod k$?

4. Separation Results

4.1. *Introduction*

In this section, we consider the question of which classes of **OptP** functions are provably distinct under the assumption that $\mathbf{P} \neq \mathbf{NP}$, 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.

For example, TRAVELING SALESPERSON is complete for $\mathbf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ and CLIQUE is complete for $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$. Also, BIN PACKING (given n items with (integral) sizes $s_1, \ldots, s_n$ and integer B, what is the fewest number of bins, each of size B, needed to hold all of the items?), using the algorithm of Karmarkar and Karp [14], can be approximated within an additive constant of $O(\log^2 n)$. The exact optimal number of bins can then be found using only $2 \log \log n + O(1)$ queries and hence BIN PACKING is in $\mathbf{FP}^{\mathrm{SAT}}[O(\log \log n)]$.

We would thus like to say that TRAVELING SALESPERSON 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 $\mathbf{P}^{\mathrm{SAT}}$ collapses to just $\mathbf{P}^{\mathrm{SAT}}[1]$.

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

Proof. Pick an oracle A such that $\mathbf{P}^A \neq \mathbf{NP}^A = \mathbf{coNP}^A$ [2]. Then, for oracle A, $\mathbf{NP}^A = \mathbf{coNP}^A$ implies that the polynomial hierarchy [21] collapses to $\mathbf{NP}^A$ and hence $\mathbf{NP}^A = \mathbf{P}^{\mathrm{SAT}^A}$ and hence $\mathbf{P}^{\mathrm{SAT}^A}[1] = \mathbf{P}^{\mathrm{SAT}^A}$. ∎

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 in this measure.

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

Proof. Assume $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)] = \mathbf{FP}^{\mathrm{SAT}}[n^{O(1)}]$. Then we show that $\mathbf{P} = \mathbf{NP}$ by showing how to recognize SATISFIABILITY in polynomial time. By hypothesis, $\mathrm{MSA} \in \mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$, so there is a $\mathbf{P}^{\mathrm{SAT}}$ machine, M, that computes MSA and makes at most $O(\log n)$ queries. Then, to determine if $\varphi \in \mathrm{SAT}$, 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 $\varphi \in \mathrm{SAT}$. ∎

We also prove a more general separation result: $\mathbf{FP}^{\mathrm{SAT}}[f(n)]$ is properly contained in $\mathbf{FP}^{\mathrm{SAT}}[g(n)]$ whenever $f(n) < g(n)$ and $f(n) \leqslant \frac{1}{2}\log n$. A corollary of this result is that CLIQUE is harder than BIN PACKING in this measure.

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

Proof. Assume that $\mathbf{FP}^{\mathrm{SAT}}[f(n)] = \mathbf{FP}^{\mathrm{SAT}}[g(n)]$, where f and g are as above. Without loss of generality, we may as well assume that $g(n) = f(n) + 1$. To show $\mathbf{P} = \mathbf{NP}$, we give an algorithm to test for SATISFIABILITY.

Define the function $\mathrm{CIRCUIT}_{g(n)}$ that takes a boolean circuit $C(x_1, ..., x_n)$ as input, and outputs the lexicographically maximum $x_1 \cdots x_{g(|C|)}$ that can be extended to a satisfying assignment for C (this is the same problem as $\mathrm{LEX}_{g(n)}$ except for circuits). Then, $\mathrm{CIRCUIT}_{g(n)} \in \mathbf{FP}^{\mathrm{SAT}}[g(n)]$ and so, by hypothesis, there is a $\mathbf{P}^{\mathrm{SAT}}$ machine, M, that computes $\mathrm{CIRCUIT}_{g(n)}$ and only uses $f(n)$ oracle queries. Using M, we will be able to eliminate inputs to the circuit until we can test for satisfiability directly.

Let C be a circuit of size n with inputs $x_1, x_2, ...$, and let $m = f(n)$. First we describe how to replace C with another circuit C' with one fewer input, where $|C'| \leqslant n + n^{2/3}$, and such that C is satisfiable if and only if C' is satisfiable. Run M on C for all possible sequences of oracle answers. This can be done in polynomial time because $f(n) < \log n$. Each sequence of oracle answers produces an assignment to $x_1, ..., x_{m+1}$; thus, M defines a function from $\{0,1\}^m \to \{0,1\}^{m+1}$. We can express this function by a boolean circuit D with m inputs and $m+1$ outputs and size $O(m^2 2^m)$ just by enumerating all possible combinations of the inputs. Since $m \leqslant \frac{1}{2}\log n$, this has size at most $O(\sqrt{n}\log^2 n) \leqslant n^{2/3}$, for sufficiently large n. Construct C' by replacing C's inputs $x_1, ..., x_{m+1}$ with the output of D. Then, C' has one fewer input than C, and the size of C' is at most $n + n^{2/3}$. Also, if C is satisfiable, then C has a maximum satisfying assignment and hence C' is satisfiable. And because the output of D still goes into the original inputs for C, if C' is satisfiable then C is satisfiable.

Now we can use this iterative step to test C for satisfiability. Let $C_0 = C$ and perform the above iteration on C_i to obtain C_{i+1}, thus eliminating one input to the circuit at each iteration, until we obtain a circuit with at most $O(\log n)$ inputs. This circuit can then be tested for satisfiability directly by trying all possible inputs.

The last thing to check is the size of the circuits. The ith iterative step produces a

circuit C_{i+1} of size $|C_i| + |C_i|^{2/3}$, where $|C_0| = n$. We claim inductively that all of these circuits have size at most n^4. If so, then the amount of increase in size in any one step is at most $(n^4)^{2/3} = n^{8/3}$. Since we perform at most n iterations and since we start with a formula of size n, the last (and largest) circuit has size at most $n + n \cdot n^{8/3} < n^4$. This proves the induction hypothesis, and hence no circuit is larger than n^4. Thus the test for SATISFIABILITY is in polynomial time. ∎

4.3. *Applications*

This last separation result has applications to approximation algorithms for **OptP** problems. If Π is an **OptP** problem and if, in polynomial time, we can approximate Π within an additive constant of $c(n)$, then we can compute the optimal value of Π with $\lceil \log c(n) \rceil$ queries by binary search. Thus, an **OptP**-completeness result for Π gives a lower bound on how closely Π can be approximated.

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

Proof. Theorem 4.2 implies that $\text{LEX}_f \notin \mathbf{FP}^{\text{SAT}}[f(n) - 1]$, unless $\mathbf{P} = \mathbf{NP}$. (This result holds for $f(n) \leq \frac{1}{2} \log n$, but since we will be stretching the input by n^k anyway, we need only assume that $f \in O(\log n)$.) Now, if Π is hard for $\mathbf{OptP}[f(n)]$, 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 $\varepsilon < 1/k$, if a polynomial-time algorithm A could approximate Π within an additive constant of $\frac{1}{2} 2^{f(n^\varepsilon)}$, then we could solve Π with $f(n^\varepsilon) - 1$ queries, and hence we could solve LEX_f with only $f(n^{\varepsilon k}) - 1$ queries. This is a contradiction to Theorem 4.2 unless $\mathbf{P} = \mathbf{NP}$. ∎

5. DISCUSSION

We conclude with a discussion of possible directions for future work. The main point of this paper 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 paper 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 it was very satisfying to discover that not all problems were the same in this new measure.

Another idea is to explore the connection between the classes **NP**, $\mathbf{D}^P$, Δ_2^p, # **P**, and **OptP**. We showed that under very general conditions, an **OptP**-completeness result yielded completeness results for **NP**, $\mathbf{D}^P$, 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 $\mathbf{FP}^{\mathrm{SAT}}$ other than $\mathbf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ and $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$. With the peculiar exception of BIN PACKING, almost all other natural problems seem to fall into one of these two categories. Exhibiting a natural complete problem for a different subclass would firmly establish the importance of $\mathbf{FP}^{\mathrm{SAT}}$ as a complexity measure. Gasarch [10] 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 certainly in $\mathbf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ and is hard for $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$. Also, the ideas in Theorem 4.1 can be used to show that it is not in $\mathbf{FP}^{\mathrm{SAT}}[O(\log n)]$ unless $\mathbf{P} = \mathbf{NP}$. But the precise complexity within these bounds of finding the clique is not known.

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

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 [23] 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.

Lastly, one could consider an **NP** metric Turing machine to allow for alternating MAX and MIN functions. The author can show that this generalization does define an interesting class of functions with natural complete problems [17]. Furthermore, k

alternations of MAX and MIN correspond to Δ_{k+1}^p in the polynomial hierarchy in a manner analogous to the correspondence between **OptP** and Δ_2^p. These ideas can also be used to show a natural complete problem for Δ_3^p.

ACKNOWLEDGMENTS

I wish to thank David Shmoys and my advisor Vijay V. Vazirani for helpful discussions, and Steve Mahaney, John Gilbert, and the referees for useful comments. Howard Karloff pointed out a simplification in the proof for COLORING, and Craig Rich pointed out an error in Theorem 3.3.

REFERENCES

1. A. AHO, J. HOPCROFT, AND J. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, MA, 1974.
2. T. BAKER, J. GILL, AND R. SOLOVAY, Relativizations of the P = ? NP question, *SIAM J. Comput.* **4** (1975), 431–442.
3. R. BEIGEL, "Query-Limited Reducibilities," Ph. D. thesis, Dept. of Computer Science, Stanford University, 1986.
4. L. BERMAN AND J. HARTMANIS, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* **6** (1977), 305–327.
5. A. CHANDRA, D. KOZEN, AND L. STOCKMEYER, "Alternation," *J. Assoc. Comput. Mach.* **28** (1981), 114–133.
6. C. CHRISTOPHIDES, "Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem," Tech. Report, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.
7. S. COOK, The complexity of theorem-proving procedures, *in* "Proceedings, 3rd Annu. ACM Symp. on Theory of Computing, 1971," pp. 151–158.
8. M. GAREY AND D. JOHNSON, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.
9. M. GAREY, D. JOHNSON, AND L. STOCKMEYER, Some simplified NP complete graph problems, *Theoret. Comput. Sci.* **8** (1976), 237–267.
10. W. GASARCH, "The Complexity of Optimization Functions," Tech. Report 1652, Dept. of Computer Science, University of Maryland, 1986.
11. T.-D. HUYNH, Deciding the inequivalence of context-free grammars with one-letter terminal alphabet is Σ_2^p-complete, *in* "Proceedings, 23rd Annu. IEEE Symp. of Foundations of Computer Science, 1982," pp. 21–31.
12. D. JOHNSON, The NP-completeness column: An ongoing guide, *J. Algorithms* **6** (1985), 434–451.
13. J. KADIN, "Deterministic Polynomial Time with $O(\log n)$ Queries," Tech. Report 86-771, Dept. of Computer Science, Cornell University, 1986.
14. N. KARMARKAR AND R. KARP, An efficient approximation scheme for the one-dimensional bin-packing problem, *in* "Proceedings, 23rd Annu. IEEE Symp. on Foundations of Computer Science, 1982," 312–320.
15. R. KARP, Reducibility among combinatorial problems, *in Complexity of Computer Computations* (R. Miller and J. Thatcher, Eds.), pp. 85–103, Plenum, New York, 1972.
16. M. KRENTEL, "The Complexity of Optimization Problems," Ph. D. thesis, Dept. of Computer Science, Cornell University, 1987.
17. M. KRENTEL, Generalizations of OptP to the polynomial hierarchy, in preparation.
18. C. PAPADIMITROU, On the complexity of unique solutions, *J. Assoc. Comput. Mach.* **31** (1984), 392–400.

19. C. PAPADIMITRIOU AND M. YANNAKAKIS, The complexity of facets (and some facets of complexity), *J. Comput. System Sci.* **28** (1984), 244–259.
20. S. SKIENA, "Complexity of Optimization Problems on Solitaire Game Turing Machines," M.S. thesis, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1985.
21. L. STOCKMEYER, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977), 1–22.
22. L. VALIANT, The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979), 189–201.
23. L. VALIANT AND V. VAZIRANI, NP is as easy as detecting unique solutions, *in* "Proceedings, 17th Annu. ACM Symp. of Theory of Computing, 1985."