The Complexity of Optimization Problems

Mark W. Krentel†

TR 85-719
December 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

† This work was partially funded by NSF grant DCR 85-03611

The Complexity of Optimization Problems

Mark W. Krentel†

Computer Science Deptartment
405 Upson Hall
Cornell University
Ithaca, New York 14853

Abstract

Many important problems in computer science, such as CLIQUE, COLORING, and TRAVELLING SALESPERSON, arise naturally as optimization problems. Typically one considers these problems as decision procedures, which are often in NP, and one shows intractibility by showing them NP-complete. We generalize the notion of an NP problem, in a manner analogous to Valiant's class #P, by considering the optimization version of the problem itself, and we show that this idea yields a natural class of problems that we call OptP. This class allows us to make finer distinctions on the complexity of optimization problems than is possible in NP. For example, assuming $P \neq NP$, we can show that TRAVELLING SALESPERSON is strictly harder than CLIQUE and that CLIQUE is strictly harder than BIN PACKING. We then relate OptP to the class of functions computable in polynomial time with an oracle for NP, by showing that every P^{SAT} function decomposes into an OptP function followed by a polynomial-time computation. This allows us to clear up a misconception on the role of uniqueness for the problem of UNIQUELY OPTIMAL TRAVELLING SALESPERSON as considered by Papadimitriou in the 1982 FOCS conference.

† *This work was partially funded by NSF grant BCR 85-03611*

1. Introduction

Many important problems in computer science, such as CLIQUE, COLORING, and TRAVELLING SALESPERSON, arise naturally as optimization problems. Instead of considering these problems as decision procedures, we study the complexity of the optimization problem itself, and we show that this idea forms a natural class of problems that we call OptP. An OptP function is computed by an NP machine that writes a binary number on each branch and then (magically) outputs the maximum (or minimum) of these numbers. Problems such as computing the size of the maximum clique in a graph fit naturally into this scheme: an NP machine can try all possible cliques in a graph and print the size of each one. Valiant [Va] used an analogous approach in defining the class #P by considering the plus function instead of the max function.

In section 2 we show that natural problems such as TRAVELLING SALESPERSON, 0-1 INTEGER LINEAR PROGRAMMING, and MAXIMUM SATISFYING ASSIGNMENT are complete for OptP, thus giving a precise characterization of their complexity. We also define subclasses of OptP by putting a bound on the number of bits used to express the value of a function. Then we show that MAXIMUM CNF SATISFIABILITY, CLIQUE, and COLORING are complete for the class of OptP functions whose values can be expressed with $O(\log n)$ bits. Although the OptP completeness proofs for many of these problems are straightforward extensions of their NP completeness proofs, the construction for COLORING requires stronger techniques and shows that the chromatic number of a graph actually contains more information than was previously known.

The motivation for considering the optimization version of problems is that it allows us to make finer distinctions on their complexity than is possible by considering them only as decision procedures. In section 4 we show that assuming $\mathsf{P} \neq \mathsf{NP}$, TRAVELLING SALESPERSON is strictly harder than CLIQUE and that CLIQUE is strictly harder than BIN PACKING. These distinctions

cannot be brought out by considering these problems as languages because the corresponding questions can be relativized.

A further result, discussed in section 3, is that any function computable in polynomial time with an oracle for NP can be decomposed into an OptP function followed by a polynomial-time computation. The significance of this result is that the essential difficulty of a $\mathsf{P}^{\mathrm{SAT}}$ computation lies in evaluating a related OptP function; and thus the number of bits needed to express the answer of an OptP function corresponds to the number of NP questions needed to determine its value. Furthermore, every OptP complete function gives rise to a Δ_2^p complete language. An application of this result is that the inherent complexity of UNIQUELY OPTIMAL TRAVELLING SALESPERSON, as considered by Papadimitriou [Pa], really comes from computing the length of the shortest tour. The uniqueness serves only to transform the problem to a decision procedure. Another and more natural way to make the TRAVELLING SALESPERSON problem complete for Δ_2^p is to ask the question, "Is the length of the optimal tour equivalent to 0 mod k?"

2. Optimisation Problems

In this section, we give the definitions of optimization problems and their corresponding reductions, and we show that several natural problems are complete for certain classes of optimization problems.

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 $x \in \Sigma^*$.

OptP is defined analogously to Valiant's [Va] class #P (sharp P). The value of a #P function is defined to be the number of accepting paths of an NP machine, or equivalently, the sum of the values over all of the branches. Thus, it is natural to consider other associative operators, such as the max function. Although OptP is defined in terms of the max function, we could equally have used the min function, and we will consider both maximization and minimization problems.

Many optimization problems fit naturally into this scheme. For example, problems such as finding the size of the maximum clique or the length of the shortest travelling salesperson tour in a graph can be computed by taking the maximum (or minimum) value over a set of feasible solutions that are definable by the branches of an NP machine.

The natural notion of reducibility between OptP problems is the metric reduction, which is essentially is a many-one reduction from one OptP function to another. Note that we need a many-one reduction in order to bring out distinctions such as saying that computing the size of the largest clique in a graph is harder than its decision procedure. If, for example, we considered Turing reducibility, then these two problems would be equivalent. And lastly, we note that metric reductions are closed under composition.

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 for all $x \in \Sigma^*$ we have $f(x) = T_2(x, g(T_1(x)))$.

We are now ready to show, for "sufficiently nice" bounds $z(n)$, that OptP$[z(n)]$ has complete functions; in particular, OptP and OptP$[O(\log n)]$ have natural complete problems. For many problems, we can obtain a proof of their OptP completeness by a straightforward extension of their NP completeness construction. This is not true, however, for COLORING. Karp's [Ka] reduction for k-COLORING constructs a graph which is k-colorable if some boolean formula is satisfiable and $(k+1)$-colorable if not. And similarly for the reduction to 3-COLORING: the graph that is obtained

has chromatic number 3 or 4 [GJS, AHU]. Our construction shows that the chromatic number of a graph contains much more information than just the answer to a single yes/no NP question.

We first show that the universal function, $\text{UNIV}_{z(n)}$, is complete for $\text{OptP}[z(n)]$ by a generic reduction. We say that 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 be smooth. Then, any $f \in \text{OptP}[z(n)]$ is metrically reducible to* UNIV_z.

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

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

Theorem 1: *The following functions are complete for* OptP *under metric reductions.*

WEIGHTED SATISFIABILITY
instance: CNF boolean formula φ 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

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 $AX \leq B$.

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 [Co], we can encode φ_x as a CNF formula where m is polynomial in n, the length of x.

Now, reduce x to $\Phi_x = (\varphi_x)^{2^{2n}}(y_1)^{2^{n-1}}(y_2)^{2^{n-2}}\cdots(y_n)^1$. 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. The construction given in [Pa] can be modified to give a straightforward reduction from WEIGHTED SAT to TRAVELLING SALESPERSON. □

Proof: MAXIMUM SATISFYING ASSIGNMENT. The reduction for 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 $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$, and $c_1, \ldots, c_m$. 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. □

Theorem 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

COLORING

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 [Ka] or in [AHU] 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 [GJS] and in [AHU], 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

$$V_n = \{\{i, j, k\} \mid 1 \leq i < j < k \leq n\} \quad \text{and} \quad E_n = \{(u, v) \mid u \wedge v = \emptyset\}.$$

The key property of these graphs, as proved in [GJb] 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

$G_n = H_n[G]$ 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 constuct 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 $G^* = G_1 + \cdots + G_{n+1}$, so that $\chi(G^*) = \max_i \chi(G_i)$, and let $k = \mathbf{opt}^{\mathrm{SAT}}(\Phi)$. 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

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

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. □

3. Applications to $\mathsf{P}^{\mathrm{SAT}}$ Computations

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

Definition: A function $f: \Sigma^* \to \mathbf{N}$ 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 decideable 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.

First we show that every function in $\mathsf{FP}^{\mathrm{SAT}}$ decomposes into an OptP problem followed by a polynomial-time computation, thus isolating the essential difficulty of $\mathsf{P}^{\mathrm{SAT}}$ computations. Since OptP functions are defined by NP machines, 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 $\mathsf{P}^{\mathrm{SAT}}$ 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: *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 $h: \Sigma^* \times \mathbf{N} \to \mathbf{N}$ 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 true 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 $10 \cdots 0$ will find a satisfying assignment; so, $\mathbf{opt}^N(x) \geq 10 \cdots 0$ and thus $b_1 = 1$. On the other hand, if $\varphi_1 \notin$ SAT, then no branch of the form $1 d_2 \cdots d_{z(n)}$ 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 is immediate; therefore, we can completely characterize $\mathsf{P}^{\mathrm{SAT}}$ computations, for both functions and languages, in terms of OptP.

Theorem 4: *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.*

We can apply theorem 3 to relate OptP to other complexity classes. For example, every OptP complete function gives rise to a Δ_2^p complete language, and every OptP function that is hard for OptP[2] gives rise to a D^p complete language.

Theorem 5: *Let f be in* OptP.

- *If f is complete for* OptP, *then there is a p-computable predicate P such that $\{x \mid P(x, f(x))\}$ is complete for Δ_2^p.*
- *If f is hard for* OptP[2], *then $\{x\#k \mid f(x) = k\}$ is complete for D^p.*
- *If f is hard for* OptP[1], *then $\{x\#k \mid f(x) \geq k\}$ is complete for* NP.

Proof: If f is complete for OptP, then P can simulate a Δ_2^p machine by using the output of f to compute the machine's oracle answers. □

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 UNIQUELY OPTIMAL TRAVELLING SALESPERSON [Pa]. We suggest that ODD MAXIMUM SATISFYING ASSIGNMENT be considered the canonical Δ_2^p complete language.

Theorem 6: *The following languages are complete for* $\mathsf{P}^{\mathrm{SAT}}$.

$\{\varphi(x_1, \ldots, x_n) \mid \text{max sat. assgn. to } \varphi \text{ has } x_n = 1\}$

$\{\varphi \# k \mid \text{max weight of } \varphi \text{ is equiv. to 0 mod } k\}$

$\{G \# k \mid \text{length of min TSP tour in } G \text{ is equiv. to 0 mod } k\}$

Proof: Let L be a $\mathsf{P}^{\mathrm{SAT}}$ complete language, and let M be a $\mathsf{P}^{\mathrm{SAT}}$ machine accepting L. First, define $f(x)$ = the oracle answers for $M(x)$; then define $f'(x) = 2f(x) + 1$ if $M(x)$ accepts and $2f(x)$ if $M(x)$ rejects. By computing f as the maximum over all sequences of M's oracle answers, we see, by an argument similar to theorem 3, that f is in OptP. Thus, L can be written as $\{x \mid f'(x) \text{ is odd}\}$. The result follows by the reductions given in section 2. □

Theorem 7: *The following languages are complete for* $\mathsf{P}^{\mathrm{SAT}}[O(\log n)]$.

$\{\varphi \# k \mid \text{max no. simul. sat. clauses in } \varphi \text{ is equiv. to 0 mod } k\}$

$\{G \# k \mid \text{size of } G\text{'s max clique is equiv. to 0 mod } k.\}$

4. Separation Results

In this section, we consider the question of which classes of OptP functions are provably distinct under the assumption that $\mathsf{P} \neq \mathsf{NP}$. Clearly, if $\mathsf{P} = \mathsf{NP}$ then all OptP functions are computable in polynomial time. We would like to say, for example, that since TRAVELLING SALESPERSON is complete for $\mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$ and since CLIQUE is in $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$, that TSP is strictly harder than CLIQUE. Unfortunately, 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 $\mathsf{P}^A \neq \mathsf{NP}^A = \mathsf{coNP}^A$ [BGS]. Then, $\mathsf{NP}^A = \mathsf{coNP}^A$ 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. We can show directly that $\mathsf{P} \neq \mathsf{NP}$ implies that n queries are strictly more powerful than $O(\log n)$ queries. This result shows that there can be no metric reduction from TSP to CLIQUE, and hence TSP is strictly harder than CLIQUE.

Theorem 8: $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)] = \mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}] \Rightarrow \mathsf{P} = \mathsf{NP}$.

Proof: Assume that $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)] = \mathsf{FP}^{\mathrm{SAT}}[n^{O(1)}]$. Then we show that $\mathsf{P} = \mathsf{NP}$ by showing how to recognize SATISFIABILITY in polynomial time. Define the function f as: for a boolean formula $\varphi(x_1, \ldots, x_n)$, let $f(\varphi)$ = the lexicographically maximum string in $\{0,1\}^n$ that satisfies φ, if $\varphi \in$ SAT; otherwise 0, if $\varphi \notin$ SAT. Then, $f \in \mathsf{OptP}$; so, by assumption, there is a $\mathsf{P}^{\mathrm{SAT}}$ machine, M, that computes f 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 can 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 $f(n) \leq \frac{1}{2}\log n$. We can use this result to show that CLIQUE is harder than BIN PACKING. Karmarkar and Karp [KK] showed 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 $O(\log \log n)$ queries to SAT, and so BIN PACKING is in $\mathsf{FP}^{\mathrm{SAT}}[O(\log \log n)]$. CLIQUE, on the other hand, is complete for $\mathsf{FP}^{\mathrm{SAT}}[O(\log n)]$.

Theorem 9: *Let f and g be smooth where $f(n) < g(n)$ and $f(n) \leq \frac{1}{2}\log n$. Then,* $\mathsf{FP}^{\mathrm{SAT}}[f(n)] = \mathsf{FP}^{\mathrm{SAT}}[g(n)] \Rightarrow \mathsf{P} = \mathsf{NP}$.

Proof: Assume f and g are as above and that $\mathsf{FP}^{\mathrm{SAT}}[f(n)] = \mathsf{FP}^{\mathrm{SAT}}[g(n)]$. We show how to recognize SATISFIABILITY in polynomial time. Define the $\mathsf{OptP}[g(n)]$ function $h(\varphi(x_1, \ldots, x_n))$ = the lex. max $x_1 \cdots x_{g(n)}$ that can be extended to a satisfying assignment. By hypothesis, $h \in \mathsf{FP}^{\mathrm{SAT}}[f(n)]$, so let M be a $\mathsf{P}^{\mathrm{SAT}}$ machine that computes h with only $f(n)$ oracle queries.

To test for satisfiability, let $\varphi(x_1, \ldots, x_n)$ be a boolean formula of length 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 $\leq g(n) \cdot 2^{f(n)} \cdot \log n \leq n$. 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. □

5. Discussion

In this paper, we considered the optimization version of problems, and we showed that this idea forms a natural class of functions with complete problems. We then showed that OptP functions captured the essential difficulty of $\mathrm{P}^{\mathrm{SAT}}$ computations, and lastly, we considered the question of which classes of OptP functions were provably harder than other classes.

This work gives rise to several new open problems. First, are there other associative operators besides the max and plus functions that define interesting classes of problems when applied to the branchs of an NP machine? Also, we could define a hierarchy of functions based on alternating max and min functions (or other functions), for which OptP is the base case. Are there natural problems complete in such a hierarchy?

Another direction to pursue is determining the precise complexity of BIN PACKING. We know that an instance of BIN PACKING contains the answer to at least one NP question and at most

$O(\log \log n)$ questions. The precise complexity of BIN PACKING, however, is not known. And finally, we would like to see other functions $z(n)$ besides n and $\log n$ for which $\mathsf{FP}^{\mathrm{SAT}}[z(n)]$ has complete problems: BIN PACKING is a possibility.

6. Acknowledgements

We would like to thank Neil Immerman for his suggestions for further work, and especially Vijay Vazirani for his continued support and encouragement.

7. References

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

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

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

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

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

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

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

[Ka] 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.

[KK] 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.

[Pa] Papadimitriou, C., "On the Complexity of Unique Solutions," *Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science*, pp. 14-20, 1982.

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