OptP as the Normal Behavior of NP-Complete Problems

William I. Gasarch[*]
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Mark W. Krentel[†]
Department of Computer Science
Rice University
Houston, Texas 77251

Kevin J. Rappoport[‡]
Department of Computer Science
University of Maryland
College Park, Maryland 20742

November 25, 1992

Abstract

In previous work [Kre88, Gas86], the authors considered the functional versions of NP problems, defined OptP as a class of functions computing the optimal value of an NP problem and gave natural complete functions. This approach has the advantage of retaining more of a problem's original structure and of unifying NP with the closely-related complexity classes D^p and Δ_2^p. The central thesis of this paper is that an NP-complete problem can usually be extended to an OptP-complete function. In support of our claim, we show how to do this for forty problems from Garey and Johnson [GJ79].

[*]Supported in part by NSF grants CCR-88-03641 and CCR-90-20079.

[†]Supported in part by NSF grant CCR-88-09370.

[‡]Author's current address: Supercomputing Research Center, 17100 Science Drive, Bowie, Maryland 20715.

1 Introduction

Many NP problems such as TRAVELING SALESMAN and CLIQUE naturally arise as optimization problems. In studying their complexity, these problems are usually first converted to an equivalent yes/no question, and this is the way that complexity classes are normally defined. Krentel [Kre88] and Gasarch [Gas86] considered the functional versions of problems, defined OptP as a class of functions computing the optimal value of an NP problem, and gave natural complete functions such as TRAVELING SALESMAN. This approach has the advantage of retaining more of the original structure in a problem and unifies completeness results for NP with other closely-related classes such as D^p and Δ_2^p. The central thesis of this paper is that if an NP-complete problem arises naturally as an optimization problem, then its functional version is also complete for OptP or a subclass of OptP, and that this is the normal behavior of NP-complete problems. In support of our claim, we show that this is the case for several problems from Garey and Johnson [GJ79].

Although NP is defined as a class of yes/no languages, many of these problems are taken from optimization problems, and we wish to study their original functional versions. For example, we are interested in computing the *length* of the shortest traveling salesman tour, the *size* of the largest clique, etc. We define OptP (Optimization Polynomial Time) as a generalization of NP in order to discuss the complexity of these problems when viewed as functions. A function is in OptP if it can be computed as the maximum (or minimum) value over the set of feasible solutions of an NP machine. For example, the length of the minimum traveling salesman tour naturally fits this definition because we can try all possible tours with an NP machine and then take the shortest one. The size of a graph's largest clique, its minimum chromatic number, and the fewest number of vertices in a vertex cover are also examples of OptP functions.

The original motivation for OptP was the observation that NP-complete problems can express much more about other problems than just their yes/no value. Often the value of the optimal solution is either preserved or closely-related in the reduction. For example, in Karp's [Kar72] reduction from SATISFIABILITY to CLIQUE, the number of vertices in the largest clique is the maximum number of simultaneously satisfiable clauses. Or, a graph with n nodes has an independent set of size k if and only if it has a vertex cover of size $n - k$. Even in Aho, Johnson and Ullman's [AJU77] reduction from FEEDBACK VERTEX SET to CODE GENERATION, the length of the optimal program is the size of the minimal feedback vertex set plus some constant. In order to capture this additional structure, we say that f is *metrically reducible* to g if given x we can compute some y such that $f(x)$ can be computed from $g(y)$ in polynomial time. Note that we only require that $f(x)$ be easily computable from $g(y)$, not that they are equal. We also say that g is OptP-complete if $g \in$ OptP and all $f \in$ OptP are metrically reducible to g. It turns out

that many NP-complete problems, when considered as functions, are also OptP-complete. For example, TRAVELING SALESMAN, GRAPH PARTITIONING, SUBSET SUM and ZERO-ONE LINEAR PROGRAMMING are all OptP-complete. This says that these problems can express the optimal value of any problem in OptP. Skiena [Ski85] defined solitaire game Turing machines and the class SGP, identical to our notion of OptP, and independently proved several complete problems.

Some problems in OptP such as CLIQUE and COLORING do not appear to be complete because their range of values is too limited. For example, a graph with n nodes cannot possibly have a clique larger than n, and this isn't enough information to express an arbitrary value with polynomially many bits. For these problems, we define OptP$[l(n)]$ to be a subclass of OptP where $f \in$ OptP$[l(n)]$ if the binary representation of $f(x)$ requires at most $l(|x|)$ bits. CLIQUE, VERTEX COVER and COLORING are all in OptP$[O(\log n)]$ and this is the natural subclass for these unweighted problems. Indeed, these three problems are complete for OptP$[O(\log n)]$ and they cannot be complete for OptP unless P = NP. So, when viewed as functions, it turns out that NP-complete problems are not all equivalent.

Another approach to OptP functions is determining the number of NP queries needed to compute the function. Let $\mathrm{F}\Delta_2^p$ be the class of functions computable in polynomial time with an oracle for NP, and let $\mathrm{F}\Delta_2^p[l(n)]$ be the subclass restricted to $l(n)$ oracle queries. The main result from [Kre88] is that any $\mathrm{F}\Delta_2^p[l(n)]$ function is metrically reducible to an OptP$[l(n)]$ function, and in particular, the correct sequence of oracle answers in a $\mathrm{F}\Delta_2^p[l(n)]$ computation is an OptP$[l(n)]$ function. A corollary of this result is that any function complete for OptP$[l(n)]$ is also complete for $\mathrm{F}\Delta_2^p[l(n)]$. This gives a way of identifying "how much" NP-completeness is embedded in OptP functions. This further says that what is hard about an $\mathrm{F}\Delta_2^p$ function is a closely-related OptP function. For example, the difficulty of Papadimitriou's [Pap84] UNIQUELY OPTIMAL TRAVELING SALESMAN problem is in computing the length of the optimal tour.

It is interesting to note that different NP-complete problems have widely varying "amounts" of NP-completeness in this measure. Many weighted problems such as TRAVELING SALESMAN, SUBSET SUM and GRAPH PARTITIONING are OptP-complete and hence also $\mathrm{F}\Delta_2^p$-complete. And many unweighted problems such as CLIQUE, COLORING and VERTEX COVER are complete for OptP$[O(\log n)]$ and $\mathrm{F}\Delta_2^p[O(\log n)]$. If we modify CLIQUE or VERTEX COVER by putting weights on the vertices and ask for the maximum weight clique, or minimum weight vertex cover, then these problems become complete for OptP and $\mathrm{F}\Delta_2^p$. Other problems are in even lower subclasses of $\mathrm{F}\Delta_2^p$. The Karmarkar and Karp algorithm [KK82] approximates BIN PACKING within an additive constant of $O(\log^2 n)$. This implies that the optimal number of bins can be computed in $\mathrm{F}\Delta_2^p[O(\log\log n)]$ by performing binary search on the remaining interval. Another example is CHROMATIC INDEX, the fewest number of colors needed to color the edges of a graph. Vizing's The-

orem [Ber85] says that the chromatic index of a graph is always h or $h+1$, where h is the degree of the graph, and Holyer [Hol81] proved that distinguishing these two cases is NP-complete. Together these results imply that CHROMATIC INDEX is $\mathrm{F}\Delta_2^p[1]$-complete. We point out that for technical reasons, CHROMATIC INDEX is in $\mathrm{OptP}[O(\log n)]$ but not in OptP[1] because the output cannot be described with a single bit. However, EXCESS CHROMATIC INDEX, the chromatic index of a graph minus its degree, *can* be described with a single bit and is OptP[1]-complete as a minimization function.

A further advantage of OptP is that it unifies NP with D^p and Δ_2^p. A language L is in D^p if it can be written as $L = L_1 \cap \overline{L}_2$ for some L_1 and L_2 in NP. Papadimitriou and Yannakakis [PY84] defined this class and showed examples of natural complete problems such as polytope facets, exact problems and critical problems. For our purposes, the most important of these examples are the exact problems, that is, asking if the value of some function is *exactly* equal to k, instead of greater than or equal to k. EXACT CLIQUE and EXACT TRAVELING SALESMAN are examples of exact problems that are complete for D^p. The advantage of the functional approach is that these results all follow from a single OptP-completeness result. In fact, under very general conditions, any function in OptP that is hard for at least OptP[2] will have its exact problem D^p-complete, and this will be true for all of our problems in Section 3.

OptP is also closely-related to Δ_2^p. A language is in Δ_2^p if it can be solved in polynomial time with an oracle for NP. This class has the same computational power as $\mathrm{F}\Delta_2^p$ but it is restricted to yes/no problems. There are only a few examples of natural complete problems for Δ_2^p. Papadimitriou [Pap84] showed the first such example: UNIQUELY OPTIMAL TRAVELING SALESMAN, that is, given an instance of traveling salesman, is the optimal solution unique. In his construction, it is crucial that the correct sequence of oracle answers in a Δ_2^p computation can be embedded in the length of the optimal tour. This is not surprising, because it is exactly this embedding that allows us to reduce any $\mathrm{F}\Delta_2^p$ function to an OptP function, and it says that the difficulty of TRAVELING SALESMAN is in computing the cost of the optimal tour. By asking a different question about the optimal solution, we can make other Δ_2^p-complete problems. Again, under very general conditions, if f is an OptP-complete function, then determining if $f(x)$ is equivalent to k_1 mod k_2 will be Δ_2^p-complete. In summary, if f is OptP-complete, then deciding if $f(x)$ is greater than (or less than) k is NP-complete, deciding if $f(x)$ is exactly equal to k is D^p-complete, and deciding if $f(x)$ equals k_1 mod k_2 is Δ_2^p-complete, and these results all follow as corollaries of the OptP-completeness result.

We claim as the central thesis of this paper that for NP-complete problems which naturally arise as optimization problems, their functional version is complete for OptP or a subclass of OptP, and that this is the normal behavior of NP-complete problems. In support of our claim, we show that this is the case for over forty problems, mainly from

Garey and Johnson [GJ79], and including INDUCED SUBGRAPH WITH PROPERTY Π from Lewis and Yannakakis [LY80]. We also have at least one problem from each section in the Appendix of [GJ79]. We don't intend our list to be exhaustive; rather, we have tried to provide a sampling of problems from diverse areas. We view our collection of problems as evidence that OptP is an important and typical aspect of NP-complete problems, and we encourage other authors to state their results in terms of OptP-completeness.

In Section 2, we review the definitions and basic results of OptP, $\mathrm{F}\Delta_2^p$, completeness, etc. Section 3 contains our list of complete problems. We have tried to keep the proofs as short as possible, often just claiming that an existing reduction works or can be modified to work. All of the results in Section 3 are stated as completeness for OptP or a subclass, and it is important to remember that several corollaries follow from OptP-completeness. These corollaries are discussed in Section 2.2. For weighted problems, there is an obvious generalization of the problem to arbitrary-length weights. That is, we restrict the length of the weights in the problem instance to at most $l(n)$ bits and often the resulting problem is OptP$[l(n)]$-complete. Since this extension straightforward for most problems, we state our results for general weights wherever possible.

2 Background

We begin with a description of the complexity classes that we will need, especially OptP and $\mathrm{F}\Delta_2^p$. Then we discuss the corollaries that follow from an OptP-completeness result. Recall that in the next section, we state all of our results as completeness for OptP or one of its subclasses. We assume some familiarity with P, NP and reduction: [GJ79] is a good reference.

2.1 Definitions

The definitions of metric Turing machine and OptP generalize NP to functions and are intended to capture our intuitive notion of a discrete optimization problem. The *size* of the largest clique in a graph or the *length* of the shortest traveling salesman tour are examples of OptP functions. Although we define OptP as a class of maximization functions, we also consider OptP to include minimization problems.

Notation $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers; Σ^* is the set of finite-length strings over some alphabet; we often write 1^k to mean a string of 1's k long; and $\log n$ means logarithm base 2.

Definition An NP *metric Turing machine*, N, is a nondeterministic polynomially time bounded Turing machine such that every branch writes a natural number in binary and

accepts. For $x \in \Sigma^*$ we write $\mathbf{opt}^N(x)$ for the maximum (or minimum) value on any branch of $N(x)$.

Definition A function $f : \Sigma^* \to \mathbb{N}$ 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 $x \in \Sigma^*$; and f is in OptP$[l(n)]$ if, in addition, the length of $f(x)$ in binary is at most $l(|x|)$ bits.

Note that OptP = OptP$[n^{O(1)}]$. We also need to consider problems computable with an NP oracle; Δ_2^p is the language class and FΔ_2^p is the function class. It is straightforward to show that OptP $\subseteq$ FΔ_2^p by binary search on the optimal value. Later we show that any FΔ_2^p function can be reduced to an OptP function. The main difference between these two classes is that an FΔ_2^p function can compute its answer by any polynomial-time function of its oracle answers, while OptP is restricted to the maximization operator. For example, the low-order bits of the length of the shortest traveling salesman tour is easily computable in FΔ_2^p but does not appear to be in OptP.

Definition A function $f : \Sigma^* \to \mathbb{N}$ is in FΔ_2^p if f is computable in polynomial time with an oracle for NP; and f is in F$\Delta_2^p[l(n)]$ if, in addition, $f(x)$ is computable with at most $l(|x|)$ queries to its oracle.

Definition A language $L \subseteq \Sigma^*$ is in Δ_2^p if L is decidable in polynomial time with an oracle for NP; and L is in $\Delta_2^p[l(n)]$ if, in addition, L is decidable with at most $l(n)$ queries to its oracle. $\Theta_2^p = \Delta_2^p[O(\log n)]$, and L is in D^p if $L = L_1 \cap \overline{L}_2$ where L_1 and L_2 are in NP.

Again, $\Delta_2^p = \Delta_2^p[n^{O(1)}]$ and FΔ_2^p = F$\Delta_2^p[n^{O(1)}]$. Among the language classes, Wagner [Wag88] studies Θ_2^p and shows that several different characterizations are actually equivalent. Θ_2^p has enough computational power to compute the optimal value of unweighted problems. For example, determining if the largest clique in a graph has an odd number of vertices is Θ_2^p-complete. Papadimitriou and Yannakakis [PY84] define D^p and show several complete languages.

For reductions between languages, we use many-one polynomial-time reductions. For functions, we use a generalization of many-one reductions called *metric reductions*. Essentially, f is reducible to g as functions if for every x we can compute a single y such that $f(x)$ can be computed from $g(y)$. We don't require that $f(x)$ equal $g(y)$, but if this is the case, then the reduction is called *exact*. If the map from $g(y)$ to $f(x)$ is a linear function, then the reduction is called *linear*. Linear reductions are an important special case because they imply several other completeness results as corollaries. All of the reductions in Section 3 are linear.

Definition Let $f, g : \Sigma^* \to \mathbb{N}$. 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 $T_2 : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Sigma^*$. A metric reduction is *linear* if for all $x \in \Sigma^*$ the map $k \mapsto T_2(x, k)$ is a linear function (but the coefficients may depend on x), and a metric reduction is *exact* if $T_2(x, k) = k$.

Definition For either functions or languages, we say that f is *hard* for a complexity class $\mathcal{C}$ if all $g \in \mathcal{C}$ are reducible to f, and f is *complete* for $\mathcal{C}$ if, in addition, $f \in \mathcal{C}$.

When considering the subclasses OptP$[l(n)]$ and F$\Delta_2^p[l(n)]$, it is convenient to impose some form of constructibility assumption on $l(n)$. We usually require that $l(n)$ be constructible and monotonic and we call this *smooth*. The hypothesis of *uniformly smooth* is used for weighted problems when restricting the weights to at most $l(n)$ bits. We discuss this in the beginning of Section 3.

Definition A function $l : \mathbb{N} \to \mathbb{N}$ is *constructible* if $1^n \mapsto 1^{l(n)}$ is computable in polynomial time, l is *smooth* if, in addition, $x \leq y$ implies $l(x) \leq l(y)$, and l is *uniformly smooth* if, in addition, there is some constant c such that $l(cn) \geq l(n) + 1$.

2.2 Corollaries

In this section, we review the corollaries that follow from an OptP-completeness result. Rather than repeat these results for each problem in the next section, we present them just once and then state all of the results in Section 3 as OptP$[l(n)]$-completeness for an appropriate $l(n)$. We emphasize that these corollaries apply to all of the problems in the next section, and indeed one of the main motivations for OptP is that these results all follow from a single completeness result. Unless otherwise noted, the theorems in this section are from [Kre88].

The first and most basic corollary is that an OptP-complete function can express the optimal value of any function in OptP. This is, of course, merely a restatement of the definition of completeness. For example, given an instance of an OptP problem, we can construct a single instance of TRAVELING SALESMAN such that the optimal value in the original problem can easily be computed from the length of the minimum traveling salesman tour. A similar result applies to CLIQUE for any OptP$[O(\log n)]$ function. These results show that TRAVELING SALESMAN and CLIQUE can express much more about an NP problem than just the answer to its yes/no question.

OptP is also closely related to FΔ_2^p. It turns out that every function in FΔ_2^p can be reduced to an OptP function. This says that the inherent difficulty of an FΔ_2^p computation is embedded in an OptP problem and that an OptP-complete function is also complete for FΔ_2^p.

Theorem 2.1 *For any constructible $l(n)$,*

1. $\mathrm{OptP}[l(n)] \subseteq \mathrm{F}\Delta_2^p[l(n)]$.
2. *Any $f \in \mathrm{F}\Delta_2^p[l(n)]$ can be expressed as $f(x) = h(x, g(x))$ where $g \in \mathrm{OptP}[l(n)]$ and $h : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ is computable in polynomial time.*

Theorem 2.2 *For any smooth $l(n)$,*

1. *$f \in \mathrm{F}\Delta_2^p[l(n^{O(1)})]$ if and only if f is metrically reducible to some $g \in \mathrm{OptP}[l(n)]$.*
2. *If f is complete for $\mathrm{OptP}[l(n)]$, then f is also complete for $\mathrm{F}\Delta_2^p[l(n)]$.*

We should point out that a metric reduction can stretch the size of a problem instance by any polynomial amount. This implies that a function in $\mathrm{F}\Delta_2^p[l(n^{O(1)})]$ can always be reduced to a function in $\mathrm{F}\Delta_2^p[l(n)]$ and therefore an $\mathrm{F}\Delta_2^p[l(n)]$-complete function is also complete for $\mathrm{F}\Delta_2^p[l(n^{O(1)})]$.

The connection with $\mathrm{F}\Delta_2^p$ identifies how many NP queries are needed to compute an OptP function. A completeness result for $\mathrm{F}\Delta_2^p[l(n)]$ says that exactly $l(n)$ queries to an NP oracle are needed to compute the function. For example, TRAVELING SALESMAN is complete for $n^{O(1)}$ queries and CLIQUE is complete for $O(\log n)$ queries. A further consequence of $\mathrm{F}\Delta_2^p$-completeness is that we can embed several yes/no questions into the value of a single $\mathrm{F}\Delta_2^p$ function. For example, given boolean formulas x_1, ..., x_n, we can construct a single instance of TRAVELING SALESMAN such that the answers to all of the questions "Is $x_i \in$ SAT?" can be easily computed from the length of the minimum traveling salesman tour. A similar result applies to CLIQUE if we restrict the number of queries to $\log n$.

Theorem 2.3 *For any smooth $l(n)$, the following function is in $\mathrm{F}\Delta_2^p[l(n)]$ and hence can be reduced to any $\mathrm{OptP}[l(n)]$-complete function.*

- QUERY$[l(n)]$
  Instance: $1^n \# x_1 \# \cdots \# x_k$ such that $|x_i| \leq n$ and $k \leq l(n)$.
  Output: $b_1 \cdots b_k$ where b_i is 1 if $x_i \in$ SAT and 0 if $x_i \notin$ SAT.

We should point out that QUERY$[l(n)]$ is probably not complete for $\mathrm{F}\Delta_2^p[l(n)]$ because the successive questions to SAT do not depend on the answers to the previous questions. The following hierarchy results are known for $\mathrm{F}\Delta_2^p$. Part (2) is from [ABG90].

Theorem 2.4 *Let $l(n)$ be smooth and assume $\mathrm{P} \neq \mathrm{NP}$.*

1. *$\mathrm{F}\Delta_2^p[l(n) - 1] \neq \mathrm{F}\Delta_2^p[l(n)]$ whenever $l(n) \leq (1 - \epsilon) \log n$ for some fixed $\epsilon > 0$.*

2. $\mathrm{F}\Delta_2^p[l(n)-1] \neq \mathrm{F}\Delta_2^p[l(n)]$ *whenever* $l(n) \leq c \log n$ *for some fixed* c *if, in addition,* $\Sigma_3^p \neq \Pi_3^p$.

3. $\mathrm{F}\Delta_2^p[O(\log n)] \neq \mathrm{F}\Delta_2^p$.

A further corollary is that every OptP-complete function can be modified to give complete languages. In particular, there is always some predicate on the optimal value that gives a Δ_2^p-complete language. This is not surprising because we already knew that the function itself was complete for $\mathrm{F}\Delta_2^p$. If f is complete under linear reductions, then there are specific questions about $f(x)$ that are complete for Δ_2^p, D^p and NP. For example, the EXACT PROBLEM for f (given x and k, is $f(x) = k$?) is D^p-complete whenever f is $\mathrm{OptP}[l(n)]$-complete under linear reductions and $l(n) \geq 2$.

Theorem 2.5 *For any smooth* $l(n)$*, if* f *is complete for* $\mathrm{OptP}[l(n)]$*, then there is some polynomial-time computable predicate* P *such that* $L = \{x\#y \mid P(x, f(y))\}$ *is complete for* $\Delta_2^p[l(n)]$*. In particular, if* f *is* OptP *or* $\mathrm{OptP}[O(\log n)]$*-complete, then* L *is* Δ_2^p *or* Θ_2^p*-complete.*

Theorem 2.6 *Let* $l(n) \geq 1$ *be smooth and let* f *be complete for* $\mathrm{OptP}[l(n)]$ *under linear metric reductions. Then,*

1. $L_1 = \{x\#k_1\#k_2 \mid f(x) = k_1 \pmod{k_2}\}$ *is* $\Delta_2^p[l(n)]$*-complete. In particular, if* f *is* OptP *or* $\mathrm{OptP}[O(\log n)]$*-complete under linear reductions, then* L_1 *is* Δ_2^p *or* Θ_2^p*-complete.*

2. $L_2 = \{x\#k \mid f(x) = k\}$ *is* D^p*-complete for* $l(n) \geq 2$.

3. $L_3 = \{x\#k \mid f(x) \geq k\}$ *is* NP*-complete (for* f *a maximization function).*

3 Completeness Results

3.1 Introduction

We begin with a discussion of the ground rules for the completeness results in this section. First of all, we have tried to keep the proofs as short as possible. Often the original NP-completeness proof is already a metric reduction or can easily be modified to give a metric reduction. In these cases, we merely cite the original construction. Whenever a problem appears in the appendix in Garey and Johnson [GJ79], we give their reference number for the problem. For example, VERTEX COVER is [GT1], the first problem in the graph theory subsection of the appendix. A summary of the reductions is in Figure 1 at the end of the paper.

We state all of the results in this section as completeness for OptP or one of its subclasses and refer the reader to Section 2.2 for a discussion of the corollaries that follow from OptP-completeness. Except for QUADRATIC PROGRAMMING, it is straightforward to verify that all of our problems are in OptP and we don't restate this fact each time.

Many problems naturally have both weighted and unweighted variations. For example, the unweighted CLIQUE function is simply the number of vertices in the largest clique in a graph, and this function is complete for OptP[$O(\log n)$]. An obvious generalization is to allow integer weights on the vertices and ask for the maximum weight of any clique, where the weight of a clique is the sum of the weights on the vertices in the clique. This weighted version of CLIQUE is complete for OptP. Furthermore, the construction for weighted CLIQUE is an easy extension to the proof of unweighed CLIQUE. In fact, for the majority of problems, it is almost no extra work to prove completeness for both weighted and unweighted versions, and we state our results in this way whenever possible. We view this behavior as further evidence that functional versions of NP-complete problems are usually also OptP-complete.

We can further generalize weighted problems by restricting the length of the weights to at most $l(n)$ bits, and this often gives an OptP[$l(n)$]-complete function. For example, with the weights on the vertices restricted to $l(n)$ bits, CLIQUE becomes OptP[$l(n)$]-complete, and again the proof is a straightforward extension of the other constructions. But this brings up a technical problem. If $x_1, \ldots, x_n$ are $l(n)$ bit integers, then $\sum x_i$ is an $l(n)+\log n$ bit integer. One solution would be to claim that the function is hard for OptP[$l(n)$] and in OptP[$l(n)+\log n$]. Instead, we require that $l(n)$ be uniformly smooth (see Section 2.1), that is, $l(cn) \geq l(n)+1$ for some c. This implies that $l(n^{O(1)}) \geq l(n)+O(\log n)$ and hence $l(n)$ grows uniformly at least as fast as $\log n$. With this assumption, we can keep the restricted weighted version in OptP[$l(n)$] by padding the size of the problem instance by some polynomial amount. In particular, any OptP[$l(n)+\log n$] function can be reduced to an OptP[$l(n^{O(1)})$] function and hence to an OptP[$l(n)$] function by padding the problem instance. This is a convenient assumption because we can add numbers at will and stay within the $l(n)$ bound.

We state our completeness results for up to three variations of each problem. In the weighted version, the weights are arbitrary integers written in binary. This version is normally complete for OptP. In the unweighted version, there are either no weights, or the weights are all 1 or from $\{0,1\}$. This version is normally complete for OptP[$O(\log n)$]. Finally, in the restricted weighted version, the weights are at most $l(n)$ bits long, where n is the size of the problem instance. This version is normally complete for OptP[$l(n)$]. Except for the generic problems, we always assume that $l(n)$ is uniformly smooth for the restricted version. When we say, for example, that a problem has "no weighted version,"

we mean either that the problem does not have a natural weighted generalization or that the weighted version is not known to be complete.

3.2 Generic Problems

In the following generic problems, the value of the OptP function can be reduced exactly.

Theorem 3.1 *For any smooth $l(n)$, the following functions are complete for* OptP$[l(n)]$ *under exact metric reductions.*

- UNIV$[l(n)]$

  Instance: $N\#x\#1^k$ where N is an NP metric Turing machine, $x \in \Sigma^*$ and $k \in \mathbb{N}$.

  Output: UNIV$[l(n)]$ simulates each branch of $N(x)$ for k moves and outputs the same value; branches of $N(x)$ that do not halt within k moves or output more than $l(|x|)$ bits have value 0.

- LEX$[l(n)]$

  Instance: Boolean formula $\Phi(x_1, \ldots, x_n)$ in conjunctive normal form and integer $m \leq l(|\Phi|)$.

  Output: The lexicographically maximum $x_1 \cdots x_m \in \{0,1\}^m$ that can be extended to a satisfying assignment, or 0 if the formula is not satisfiable.

- CIRCUIT OUTPUT$[l(n)]$

  Instance: Boolean circuit C with n inputs and m outputs where $m \leq l(|C|)$.

  Output: The lexicographically maximum output of C in $\{0,1\}^m$.

Proof: The constructions for UNIV and LEX are in [Kre88]. Given a boolean formula $\Phi(x_1, \ldots, x_n)$ and $m \in \mathbb{N}$, it is straightforward to construct a circuit $C(x_1, \ldots, x_n)$ with output $x_1 \cdots x_m$ if x_1, ..., x_n satisfies Φ and 0 otherwise (see [Lad75]). This gives an exact metric reduction from LEX$[l(n)]$ to CIRCUIT$[l(n)]$. □

3.3 SAT Variations

The main interest in the following variations of SATISFIABILITY is in proving other problems complete. CHEATING SAT has somewhat unusual constraints and is especially useful in the unweighted versions of 3-DIMENSIONAL MATCHING and TRAVELING SALESMAN. In SAT and CHEATING SAT, the constraints and the objective function play dual roles. SAT requires a legal truth assignment for the variables and then tries to maximize the number of satisfied clauses. CHEATING SAT requires that every clause be satisfied and tries to minimize the number of variables with an illegal truth assignment.

We write boolean formulas in conjunctive normal form as $\Phi = (C_1)^{w_1} \cdots (C_m)^{w_m}$ where $C_1, \ldots, C_m$ are the clauses in Φ and w_i is the weight associated with clause C_i. The weight of an assignment is the sum of the weights on the satisfied clauses. In the unweighted case, the value of an assignment is the number of satisfied clauses.

Theorem 3.2 *The weighted, unweighted and restricted weighted versions of the following functions are complete for* OptP, OptP$[O(\log n)]$ *and* OptP$[l(n)]$ *(for uniformly smooth* $l(n) \geq \log n$*) under linear metric reductions.*

- SATISFIABILITY (SAT) [LO1, LO2, LO5]

  Instance: Boolean formula $\Phi(x_1, \ldots, x_n) = C_1 \cdots C_m$ in conjunctive normal form with weights $w_1, \ldots, w_m$ on the clauses.

  Output: The maximum weight of an assignment.

  Remark: Also holds with at most two literals per clause (2-SAT).

- POSITIVE NOT-ALL-EQUAL 3-SAT [LO3]

  Instance: Same as SAT with the restrictions that every variable appears positively (un-negated) and that every clause has at most three literals.

  Output: Same as SAT except that a clause is "satisfied" if it has at least one true literal and at least one false literal.

- ONE-IN-THREE SAT [LO4]

  Instance: Same as SAT with the restriction that every clause has at most three literals.

  Output: Same as SAT except that a clause is "satisfied" if it has exactly one true literal.

- CHEATING SAT

  Instance: Same as SAT except that the weights are associated with the variables instead of the clauses.

  Output: The minimum cost of a $\{0, 1, *\}$ assignment to the variables that satisfies every clause, where $x = 1$ satisfies clauses containing x, $x = 0$ satisfies clauses containing $\overline{x}$ and $x = *$ satisfies clauses containing either x or $\overline{x}$, and the cost of an assignment is the sum of the weights of the variables assigned $*$.

  Remark: Also holds with at most two literals per clause.

Proof: The weighted version of SATISFIABILITY is shown in [Kre88] and it is straightforward to extend the result to the restricted weighted case. In all of the unweighted

versions of SAT, the weights only need to be polynomially large, so they can be removed by repeating clauses.

Let $\Phi(x_1, \ldots, x_n) = (C_1)^{w_1} \cdots (C_m)^{w_m}$ be a boolean formula in conjunctive normal form where w_j is the weight of clause C_j and let $M = \sum w_i$. We put Φ in 2CNF by making a new variable for every occurrence of every literal in Φ. If x_i or $\overline{x}_i$ appears in C_j then make a new variable x_i^j. Replace $C_j = (y_1 + \cdots + y_p)^w$ with $(y_1^j)^w \cdots (y_p^j)^w$ and add clauses $(\overline{y}_a^j + \overline{y}_b^j)^M$ for every $1 \leq a < b \leq p$ to say that a clause can only be satisfied once. Also add clauses $(\overline{y}_a^i + \overline{y}_b^j)^M$ for every pair of literals y_a in C_i and y_b in C_j such that $y_a = \overline{y}_b$ to say that we can't use both y_a and $\overline{y}_a$. Call the resulting formula Ψ. An optimal assignment to Ψ must satisfy all of the clauses with weight M, so this reduces SAT to 2-SAT.

Next, we reduce 2-SAT to POSITIVE NOT-ALL-EQUAL 3-SAT. Note that the construction for 2-SAT only uses clauses of the form (x) and $(\overline{x} + \overline{y})$. Let 0 and 1 be two new variables. Replace (x) with $\mathrm{NAE}(x, 0, 0)$, and replace $(\overline{x} + \overline{y})$ with $\mathrm{NAE}(x, y, 1)$. Finally, add a clause $\mathrm{NAE}(0, 0, 1)$ with weight greater than the sum of the weights on all of the other clauses. We then reduce this to ONE-IN-THREE SAT by replacing $\mathrm{NAE}(x, y, z)$ with $1/3(x, y, z)$ and $1/3(\overline{x}, \overline{y}, \overline{z})$.

Finally, we reduce 2-SAT to CHEATING SAT. Let $\Phi(x_1, \ldots, x_n) = (C_1)^{w_1} \cdots (C_m)^{w_m}$ be a boolean formula in 2CNF. Again, we replace occurrences of literals in Φ with new variables. Replace x_i in C_j with a new variable x_i^j, add clauses $(x_i^j + \overline{x}_i^k)(\overline{x}_i^j + x_i^k)$ for every pair $j \neq k$, and let x_i^j have weight w_j. Assigning $*$ to x_i^j corresponds to failing to satisfy C_j, so this reduces 2-SAT to CHEATING SAT. □

3.4 Graph Theory

In the following problems, we usually associate the weights with either the vertices or the edges in the graph. The weight of a subgraph is the sum of the weights on the vertices or edges in the subgraph.

CLIQUE and APPROXIMATE CLIQUE are also dual problems in how they treat the constraints and the objective function. In CLIQUE the subgraph must really be a clique and we maximize the size of the subgraph. In APPROXIMATE CLIQUE the size of the subgraph is fixed and we try to maximize the number of edges.

Theorem 3.3 *Except for* COLORING, *the weighted, unweighted and restricted weighted versions of the following functions are complete for* OptP, OptP$[O(\log n)]$ *and* OptP$[l(n)]$ *(for uniformly smooth* $l(n) \geq \log n$*) under linear metric reductions.*

- CLIQUE [GT19]

  Instance: Graph $G = (V, E)$ with weights on the vertices.

Output: The maximum weight of any clique in G.

- APPROXIMATE CLIQUE

  Instance: Graph $G = (V, E)$ with weights on the edges and integer k.

  Output: The maximum sum of the weights on the edges in an induced subgraph $V' \subseteq V$ of size exactly k.

- INDEPENDENT SET [GT20]

  Instance: Same as CLIQUE.

  Output: The maximum weight of any independent set in G.

- VERTEX COVER [GT1]

  Instance: Same as CLIQUE.

  Output: The minimum weight of any vertex cover for G.

- DOMINATING SET [GT2]

  Instance: Same as CLIQUE.

  Output: The minimum weight of any dominating set for G.

  Remark: $V' \subseteq V$ is a dominating set if for all $v \in V - V'$ there exists $v' \in V'$ such that $(v, v') \in E$.

- KERNEL [GT57]

  Instance: Directed graph $G = (V, E)$ with weights on the vertices.

  Output: The maximum weight of any kernel in G or else 0 if G has no kernel.

  Remark: $V' \subseteq V$ is a kernel if V' is an independent set and for all $v \in V - V'$ there exists $v' \in V'$ such that $(v', v) \in E$.

- LONGEST CYCLE [GT37, GT38, ND28]

  Instance: Graph $G = (V, E)$ with weights on the edges.

  Output: The length of the longest simple cycle in G.

  Remark: Holds for directed and undirected graphs, and with weights on the vertices or the edges.

- COLORING [GT4]

  Instance: Graph $G = (V, E)$.

  Output: The chromatic number of G.

  Remark: No weighted or restricted weighted version. This result immediately implies the same result for PARTITION INTO CLIQUES [GT15]. Also complete

(APPROXIMATE COLORING) if the instance includes an integer k and weights on the edges, every vertex must be assigned one of k colors and we ask for the minimum sum of the weights on the edges with both endpoints assigned the same color.

- FEEDBACK VERTEX SET [GT7]

  Instance: Directed graph $G = (V, E)$ with weights on the vertices.

  Output: The minimum weight of $V' \subseteq V$ such that $G - V'$ is acyclic.

  Remark: Also complete if we ask for the maximum weight of $V' \subseteq V$ such that the subgraph induced by V' is acyclic.

- FEEDBACK EDGE SET [GT8]

  Instance: Directed graph $G = (V, E)$ with weights on the edges.

  Output: The minimum weight of $E' \subseteq E$ such that $G - E'$ is acyclic.

  Remark: Also complete if we ask for the maximum weight of any $E' \subseteq E$ such that (V, E') is acyclic.

- INDUCED SUBGRAPH WITH PROPERTY Π [GT21]

  Instance: Graph $G = (V, E)$ with weights on the vertices.

  Output: The maximum weight of any $V' \subseteq V$ such that the subgraph induced by V' satisfies Π.

  Remark: Holds for any hereditary property Π that is true for arbitrarily large graphs, not true for all graphs and is verifiable in polynomial time. See [LY80].

Proof: CLIQUE, INDEPENDENT SET and VERTEX COVER. The reduction from SAT in [Kar72] is sufficient for the unweighted version of CLIQUE and it is straightforward to extend this result to the weighted versions. The reductions to INDEPENDENT SET and VERTEX COVER are immediate.

Proof: APPROXIMATE CLIQUE. Reduction from MAX CUT. For the weighted version, let $G = (V, E)$ be a graph where $V = \{1, 2, \ldots, n\}$, let w_{ij} be the weight on $(i, j) \in E$ or 0 if $(i, j) \notin E$ and let $M = \sum w_{ij}$. We reduce G to the graph $G' = (V', E')$ where $V' = \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ and E' includes an edge between every pair of vertices except (x_i, y_i). The weight on (x_i, x_j) is M for $i \neq j$ and similarly for the y_i's and (x_i, y_j) has weight $M + w_{ij}$ for $i \neq j$. Finally, $k = n$.

Any subset of V' with size n and only one element from each pair x_i, y_i has weight at least $\binom{n}{2}M$ and a subset with size n and both x_i and y_i for some i has weight at most $\binom{n}{2}M$. So, the optimal solution in G' must include exactly one from each pair and therefore

correspond to a valid cut in G. Thus, the optimal weight in G' is $\binom{n}{2}M$ plus the optimal weight in G.

In the unweighted case, the reduction from MAX CUT is conceptually similar to the weighted case, except that we repeat vertices instead of weighting the edges. Again, let $G = (V, E)$ be a graph with $V = \{1, 2, \ldots, n\}$ and reduce G to $G' = (V', E')$. V' includes vertices x_{ij} and y_{ij} for $1 \leq i \leq n$ and $1 \leq j \leq M$ where M is chosen later. Say that $X_i = \{x_{i1}, x_{i2}, \ldots, x_{iM}\}$ is a *group* of vertices at level i (and similarly for Y_i) and say that the x_{ij}'s are on the left *side* and the y_{ij}'s are on the right side. Within a group, E' includes all $\binom{M}{2}$ edges (x_{ij_1}, x_{ij_2}) for $j_1 \neq j_2$, and similarly for the y_{ij}'s. Across groups on the same side, E' includes all edges $(x_{i_1 j_1}, x_{i_2 j_2})$ for $i_1 \neq i_2$ except (x_{ij}, x_{ji}). This is $M^2 - 1$ edges. Across sides on the same level, E' includes *no* edges of the form (x_{ij_1}, y_{ij_2}). And across sides at different levels, E' includes all edges $(x_{i_1 j_1}, y_{i_2 j_2})$ for $i_1 \neq i_2$ except that (x_{ij}, y_{ji}) is included if and only if $(i, j) \in E$. This is either M^2 or $M^2 - 1$ edges depending on $(i, j) \in E$. Call the edge $(x_{ij}, y_{ji}) \in E'$ for $(i, j) \in E$ *special* and call the other edges *basic*. Finally, $k = nM$, half of the vertices.

Say that a subset of V' of size nM is *good* if every level i includes all of X_i and none of Y_i or vice versa. This corresponds to a valid cut in G. Also, a good subset has exactly $n\binom{M}{2} + \binom{n}{2}(M^2 - 1) = \binom{nM}{2} - \binom{n}{2}$ basic edges and any subset has at most $\binom{n}{2}$ special edges. A subset of size nM that fails to be good must have some level i with $|X_i| + |Y_i| \geq M$ and both X_i and Y_i non-empty. One of these must have size at least $M/2$ and so the subset must omit at least $M/2$ basic edges. If we take $M > 2\binom{n}{2}$, then a bad subset cannot be optimal. Thus, the optimal solution in G' corresponds to a cut in G and has size $\binom{nM}{2} - \binom{n}{2}$ plus the size of the optimal cut in G.

Proof: DOMINATING SET. Reduction from SET COVER. Let $S_1, \ldots, S_k$ be subsets of $\{x_1, \ldots, x_n\}$ and suppose S_i has weight w_i. Let $G = (V, E)$ be a graph where $V = \{S_1, \ldots, S_k, x_1, \ldots, x_n\}$. Put an edge between every S_i and S_j, put no edges between any x_i and x_j, and put an edge between x_i and S_j if $x_i \in S_j$. Let S_i have weight w_i and let each x_i have weight $\sum_{j=1}^{k} w_j$. The x_i's are too expensive to use in an minimum dominating set, so a subset of the S_i's is a dominating set in G if and only if it is a set cover.

Proof: KERNEL. Immediate reduction from INDEPENDENT SET. Replace each undirected edge (u, v) by the two directed edges (u, v) and (v, u). A kernel in the new graph corresponds to a maximal independent set in the old graph.

Proof: LONGEST CYCLE. The unweighted version is proved in [Kre88] and it is straightforward to extend the result to the weighted versions.

Proof: COLORING is in [Kre88] and APPROXIMATE COLORING is immediate from MAX CUT.

Proof: FEEDBACK VERTEX/EDGE SET. The constructions in [Kar72] are sufficient for the unweighted versions, and it is straightforward to extend the results to the weighted versions.

Proof: INDUCED SUBGRAPH WITH PROPERTY Π. We use Lewis and Yannakakis's reduction from VERTEX COVER [LY80]. In the unweighted case, Lewis and Yannakakis show that there is a vertex cover of size l if and only if deleting nkl vertices is sufficient to leave a graph that satisfies Π, so their reduction is already a linear reduction. In the weighted case, we note that the vertices c_1 and d in their construction correspond directly to the vertices in the original VERTEX COVER problem. So, it is sufficient to use their graph, carry over the weights on these vertices and put a large weight on all other vertices. □

3.5 Network Design

Theorem 3.4 *The weighted, unweighted and restricted weighted versions of the following functions are complete for* OptP, OptP[$O(\log n)$] *and* OptP[$l(n)$] *(for uniformly smooth* $l(n) \geq \log n$*) under linear metric reductions.*

- MAX CUT [ND16]

  Instance: Graph $G = (V, E)$ with weights on the edges.

  Output: The maximum weight of any partition of V into two pieces (of any size), where the weight of a partition is the sum of the weights on the edges that cross the partition.

- GRAPH PARTITIONING [ND14]

  Instance: Same as MAX CUT.

  Output: The minimum weight of a partition of V into two equal-size pieces.

  Remark: The maximization version is also complete.

- STEINER TREE [ND12]

  Instance: Graph $G = (V, E)$ with weights on the edges and subset $R \subseteq V$.

  Output: The minimum weight of a subtree of G that contains R.

- TRAVELING SALESMAN [ND22]

  Instance: Complete graph $G = (V, E)$ with weights on the edges.

  Output: The length of the shortest cycle in G that visits every vertex exactly once.

  Remark: In the unweighted version, the edge costs are from $\{1, 2\}$. All results hold with the triangle inequality.

- INTEGRAL FLOW WITH MULTIPLIERS [ND33]

  Instance: Directed graph $G = (V, E)$, distinguished vertices $s, t \in V$, multipliers $h(v) \in \mathbb{N}$ for vertices $v \neq s, t$ and capacities $c(e) \in \mathbb{N}$ for edges $e \in E$.

  Output: The maximum flow $f : E \to \mathbb{N}$ from s to t such that $0 \leq f(e) \leq c(e)$ for all $e \in E$ and $\sum_{(v,w)\in E} f(v, w) = h(v) \sum_{(u,v)\in E} f(u, v)$ for all $v \neq s, t$.

  Remark: In the unweighted version, all multipliers are from $\{0, 1, 2\}$ and all capacities are 1.

Proof: MAX CUT. Reduction from POSITIVE NOT-ALL-EQUAL 3-SAT. Replace a clause NAE(x, y, z) of weight w with a triangle between x, y and z where each edge has weight w, and replace a clause NAE(x, y) of weight w with a single edge between x and y with weight $2w$. The weight of a partition is twice the weight of an assignment.

Proof: GRAPH PARTITIONING. We can reduce MAX CUT to MAXIMUM GRAPH PARTITIONING by adding a sufficient number of isolated vertices. Then MAXIMUM GRAPH PARTITIONING can be reduced to MINIMUM GRAPH PARTITIONING by changing the weight of each edge from w to $M - w$ (considering a non-edge to have weight 0), where M is a constant larger than the weight on any edge.

Proof: STEINER TREE. We modify Karp's construction [Kar72] to give a reduction from SET COVER. We use Karp's graph but with different edge costs. Suppose S_1, ..., S_k are sets with costs c_1, ..., c_k containing elements x_1, ..., x_n. The graph has vertices $V = \{*, S_1, \ldots, S_k, x_1, \ldots, x_n\}$, edges $(*, S_j)$ for $1 \leq j \leq k$ and (S_j, x_i) if $x_i \in S_j$, and $R = \{*, x_1, \ldots, x_n\}$.

In the weighted versions, the cost of $(*, S_j)$ is c_j and the cost of (S_j, x_i) is $M = \sum c_j$. In an optimal steiner tree, every x_i must have degree one because it would be cheaper to replace an edge (S_j, x_i) with $(*, S_j)$. So the steiner tree must represent a valid set cover and will have cost nM plus the cost of the cover.

In the unweighted version, the cost of every edge is 1. Again, if some x_i has degree more than one, an edge (S_j, x_i) can be replaced with an edge $(*, S_j)$ with no increase in cost. So, the cost of the optimal steiner tree is n plus the size of the minimal cover.

Proof: TRAVELING SALESMAN. The weighted version is proved in [Kre88] and the result can be extended to the triangle inequality by adding a large constant to the cost of every edge. For the unweighted version, we first reduce CHEATING 3-SAT to DIRECTED CHEATING HAMILTON CYCLE (given a graph, find a permutation of the vertices that minimizes the number of non-edges and output the number of non-edges) by modifying the construction from SAT to HAMILTON CYCLE in [HU79] as follows. By putting all of the positive occurrences of a variable before any of its negative occurrences, the number of non-edges in CHEATING HAMILTON CYCLE is at most the number of variables cheated on in

SAT. We claim without proof that there is no more powerful way to pick non-edges. That is, any occurrence of a non-edge in HAMILTON CYCLE can be replaced by a non-edge in the gadget representing the choice for some variable. Thus, the minimum number of cheated variables is the minimum number of non-edges. Then, DIRECTED CHEATING HAMILTON CYCLE can be reduced to UNDIRECTED CHEATING HAMILTON CYCLE by the construction in [HU79] or in [Kar72]. Again, we claim without proof that any non-edge for the undirected problem can be replaced with a non-edge that corresponds to a non-edge in the directed problem.

Proof: INTEGRAL FLOW WITH MULTIPLIERS. [Sah74] gives a reduction from SUBSET SUM such that $\mathbf{opt}^{\text{SUBSET SUM}} = \mathbf{opt}^{\text{FLOW}}$ which is sufficient for the weighted version. However, this reduction is not sufficient for the unweighted or restricted weighted versions because these versions of SUBSET SUM are not complete.

For the general case, we give a reduction from 2-SAT. Let $\Phi(x_1, \ldots, x_n) = C_1 \cdots C_m$ be a boolean formula with weights $w_1, \ldots, w_m$ on the clauses. Let $G = (V, E)$ be a graph where $V = \{s, t, s_1, \ldots, s_n, x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n, c_1, \ldots, c_m, d\}$. There are edges (s, s_i), (s_i, x_i) and $(s_i, \overline{x}_i)$ with capacity 1 for each $1 \leq i \leq n$. There are edges (x_i, c_j) with capacity 1 if $x_i \in C_j$, and similarly for $(\overline{x}_i, c_j)$ if $\overline{x}_i \in C_j$. There are edges (c_j, t) and (c_j, d) with capacity w_j for each $1 \leq j \leq m$. The multiplier for x_i is the number of times x_i occurs in Φ, and similarly for $\overline{x}_i$. The multiplier for c_j is w_j, the multiplier for d is 0 and the multiplier on all other vertices is 1. In an optimal flow, either x_i or $\overline{x}_i$ will have its maximum capacity and the other will have zero flow, and the flow into c_j will be the number of true literals in C_j. The flow into t is the sum of the weights on the satisfied clauses and the excess flow drains into d. In the unweighted case, the multipliers on x_i and $\overline{x}_i$ can be replaced by several vertices with multiplier 2. □

3.6 Sets, Covers and Partitions

Theorem 3.5 *The weighted, unweighted and restricted weighted versions of the following functions are complete for* OptP, OptP[$O(\log n)$] *and* OptP[$l(n)$] *(for uniformly smooth* $l(n) \geq \log n$*) under linear metric reductions.*

- 3-DIMENSIONAL MATCHING [SP1, SP2, SP3]

  Instance: Sets W, X and Y and weighted subset $M \subseteq W \times X \times Y$.

  Output: The maximum weight of a matching $M' \subseteq M$.

  Remark: Also called EXACT COVER. Also holds if the weights are associated with the elements in W, X and Y instead of the triples in M.

- SET COVER [SP5, SP8]

Instance: Set S and weighted collection C of subsets of S.

Output: The minimum weight of $C' \subseteq C$ that covers S.

- ELEMENT COVER

  Instance: Weighted set S, collection C of subsets of S and integer k.

  Output: The maximum weight that can be covered by any $C' \subseteq C$ with $|C'| \leq k$.

Proof: 3-DIMENSIONAL MATCHING. For the weighted and restricted weighted versions, the reduction from 3-SAT in [GJ79] will work. Put a large weight on the sets covering the "internal" elements, give the sets covering the "clause" elements the weight of the clause and eliminate the "garbage collection" sets. A maximum matching must cover all of the internal elements, so it represents a legal assignment to the variables. The weight of the assignment will be the weight of the sets representing the clauses.

For the unweighted version, the construction in [GJ79] will also work as a reduction from CHEATING SAT. In the "truth-setting" component, put all positive occurrences of a variable together and all negative occurrences together. By missing only one "internal" node, a cover can get credit for the clauses containing either positive or negative occurrences of the variable. We claim without proof that this is the most powerful way to cheat.

Proof: SET COVER. Immediate reduction from VERTEX COVER.

Proof: ELEMENT COVER. Reduction from SAT. Let $\Phi(x_1, \ldots, x_n) = C_1 \cdots C_m$ be a boolean formula with weights $w_1, \ldots, w_m$ on the clauses. Let $S = \{c_1, \ldots, c_m, y_1, \ldots, y_n\}$ where c_j has weight w_j and y_i has weight $\sum w_j$. Let $\mathcal{C} = \{X_1, \ldots, X_n, \overline{X}_1, \ldots, \overline{X}_n\}$ where X_i contains y_i and all c_j such that x_i satisfies C_j and $\overline{X}_i$ contains y_i and all c_j such that $\overline{x}_i$ satisfies C_j. Finally, let $k = n$. Any optimal $\mathcal{C}' \subseteq \mathcal{C}$ with $|\mathcal{C}'| = k$ must cover $\{y_1, \ldots, y_n\}$ and so it must contain exactly one of X_i and $\overline{X}_i$. This reduces SAT to ELEMENT COVER. □

For the weighted set problems SUBSET SUM and PARTITION below, we can show OptP-completeness with general weights, but these results do not extend to the unweighted or restricted weighted versions. These problems are solvable in pseudo-polynomial time and seem to need polynomially long integers even to simulate an OptP[1] function. For example, if all of the a_i's are 1, then these problems become trivial. Or, if the weights are restricted to $l(n)$ bits, then there is a dynamic programming solution that uses time $2^{O(l(n))}$.

Theorem 3.6 *The following functions are complete for* OptP *under linear metric reductions.*

- SUBSET SUM [SP13]

  Instance: Set $A = \{a_1, \ldots, a_n\}$ of positive integers and integer K.

Output: The minimum of $|\sum_{i \in S} a_i - K|$ for any subset $S \subseteq \{1, \ldots, n\}$.

Remark: No unweighted or restricted weighted version. Also complete if we ask for the maximum $\sum_{i \in S} a_i$ less than or equal to K, or if we ask for the minimum $\sum_{i \in S} a_i$ greater than or equal to K. This implies similar results for KNAPSACK [MP9] and INTEGER EXPRESSION MEMBERSHIP [AN18].

- PARTITION [SP12]

  Instance: Set $A = \{a_1, \ldots, a_n\}$ of positive integers.

  Output: The minimum of $|\sum_{i \in S} a_i - \sum_{i \notin S} a_i|$ for any subset $S \subseteq \{1, \ldots, n\}$.

  Remark: No unweighted or restricted weighted version.

Proof: The version of SUBSET SUM in which we ask for the maximum $\sum_{i \in S} a_i$ less than or equal to K was called KNAPSACK and was shown in [Kre88]. In that reduction, the difference between $\sum_{i \in S} a_i$ and K is in its lowest order bits, and any sum greater than K must differ in higher order bits and hence is farther away from K. Thus, this reduction also works for SUBSET SUM.

We can then reduce this version to PARTITION. Let $S = \sum_{i=1}^{n} a_i$, let $\alpha = 3S$ and let $\beta = 2S + 2K$. An optimal partition to $A \cup \{\alpha, \beta\}$ must put α and β on opposite sides, and this reduces SUBSET SUM to PARTITION. □

3.7 Data Storage and Compression

Theorem 3.7 *The following functions are complete for* OptP$[O(\log n)]$ *under linear metric reductions.*

- SHORTEST COMMON SUPERSEQUENCE [SR8]

  Instance: Finite alphabet Σ and finite subset $R \subseteq \Sigma^*$.

  Output: The length of the shortest $x \in \Sigma^*$ such that every string in R is a subsequence of x.

- LONGEST COMMON SUBSEQUENCE [SR10]

  Instance: Same as SHORTEST COMMON SUPERSEQUENCE.

  Output: The length of the longest $x \in \Sigma^*$ such that x is a subsequence of every string in R.

Proof: COMMON SUB/SUPER-SEQUENCE. The reductions from VERTEX COVER in [Mai78] are linear reductions. □

3.8 Scheduling Problems

Theorem 3.8 *The following function is complete for* OptP *under linear metric reductions.*

- TWO-PROCESSOR SCHEDULING [SS8]

  Instance: Set of jobs J with integer lengths.

  Output: The minimum length of a two-processor schedule for J.

  Remark: No unweighted or restricted weighted version.

  Proof: TWO-PROCESSOR SCHEDULING. Immediate from PARTITION. □

Theorem 3.9 *The weighted, unweighted and restricted weighted versions of the following function are complete for* OptP, OptP$[O(\log n)]$ *and* OptP$[l(n)]$ *(for uniformly smooth* $l(n) \geq \log n$*) under linear metric reductions.*

- PRECEDENCE CONSTRAINED SCHEDULING [SS3, SS9]

  Instance: Set of jobs with unit runtimes, weights on the jobs, precedence relation $\prec$, number of processors p and integer deadline D.

  Output: The minimum sum of weights of the tardy jobs.

  Remark: Also complete to compute the minimum $\sum w_i C_i$ or $\sum w_i T_i$ where w_i is the weight of job J_i, C_i is the completion time for J_i and T_i is the tardiness of J_i.

Proof: PRECEDENCE CONSTRAINED SCHEDULING. We use the construction in [LK78] as a reduction from APPROXIMATE CLIQUE. Let $G = (V, E)$ be a graph, let $n = |V|$ and $m = |E|$, let w_{ij} be the weight of $(i, j) \in E$, let $M = \sum w_{ij}$ and let k be the size of the subset of V. We write the reduction for a variable number of processors. It is easy to convert this to a constant number of processors by adding dummy jobs as in [Ull75].

The set of jobs is $J = V \cup E$ and the precedence is $i, j \prec (i, j)$ where i and j are the endpoints of (i, j). The weight of $i \in V$ is M and the weight of (i, j) is w_{ij}. The number of processors at time 1 is k, at time 2 it is $\binom{k}{2} + n - k$, at time 3 it is unlimited and the deadline is $D = 2$. The optimal solution must schedule all of the vertices by the deadline, so this leaves room at time 2 for the edges among the vertices scheduled at time 1. The cost of the optimal schedule will be M minus the weight of the optimal subset of V in APPROXIMATE CLIQUE. □

Theorem 3.10 *The following function is complete for* OptP$[O(\log n)]$ *under linear metric reductions.*

- MINIMAL COMPLETION TIME [SS9]

  Instance: Same as PRECEDENCE CONSTRAINED SCHEDULING without the weights on the jobs and the deadline D.

  Output: The minimal time D such that all of the jobs can be completed by time D.

Proof: MINIMAL COMPLETION TIME. We modify the proof for PRECEDENCE CONSTRAINED SCHEDULING to give a reduction from unweighted SATISFIABILITY. In the previous construction, restrict the number of processors at time 3 to be $m - \binom{k}{2}$ and set the deadline to 3. The completion time will be 3 if and only if the graph has a clique of size k, otherwise it will be 4 (this was the NP-completeness proof of SCHEDULING in [LK78]). By combining the reductions from SATISFIABILITY to CLIQUE to SCHEDULING, we can reduce a boolean formula to a scheduling problem that has completion time 3 or 4 depending on whether or not the formula is satisfiable.

Now, let Φ be a CNF formula with m clauses. By Cook's theorem [Coo71], we can construct formulas $F_1, \ldots, F_m$ where F_i is satisfiable if and only if Φ has at least i simultaneously satisfiable clauses. By the above argument, reduce each F_i to a scheduling problem S_i such that S_i has completion time 3 or 4 depending on F_i being satisfiable. Combine $S_1, \ldots, S_m$ into a single scheduling problem $\mathcal{S}$ by extending the precedence so that all jobs in S_1 must precede all jobs in S_2, all jobs in S_2 must precede all jobs in S_3, and so on. Then the minimal completion time for $\mathcal{S}$ will be $3m$ plus the number of unsatisfied clauses in Φ. □

3.9 Mathematical Programming

Theorem 3.11 *Except as noted, the weighted, unweighted and restricted weighted versions of the following functions are complete for* OptP*,* OptP$[O(\log n)]$ *and* OptP$[l(n)]$ *(for uniformly smooth $l(n) \geq \log n$) under linear metric reductions.*

- ZERO-ONE LINEAR PROGRAMMING [MP1]

  Instance: Integer matrix A and integer vectors B and C.

  Output: The maximum value of $C^T X$ over all 0-1 vectors X satisfying $AX \leq B$, or else 0 if no X satisfies the inequality.

- QUADRATIC PROGRAMMING [MP2]

  Instance: Rational matrix A, rational vector B and rational numbers $c_1, \ldots, c_m$ and $d_1, \ldots, d_m$.

  Output: The maximum value of $\sum_{i=1}^{m}(c_i x_i^2 + d_i x_i)$ taken over rational vectors X satisfying $AX \leq B$, or else 0 if no X satisfies the inequality.

Remark: Not known to be in OptP and no unweighted version. Weighted and restricted weighted versions are OptP and OptP[$l(n)$] hard.

Proof: ZERO-ONE LINEAR PROGRAMMING. The weighted version is shown in [Kre88] and it is straightforward to extend the result to the unweighted and restricted weighted versions.

Proof: QUADRATIC PROGRAMMING. The constraint $0 \leq x \leq 1$ together with a large weight for M where $Mx(x-1)$ is in the objective function force x to be 0 or 1. The rest of the construction proceeds as in ZERO-ONE LINEAR PROGRAMMING. □

3.10 Miscellaneous Problems

Theorem 3.12 *For any uniformly smooth $l(n) \geq \log n$, the weighted, unweighted and restricted weighted versions of the following functions are complete for* OptP, OptP[$O(\log n)$] *and* OptP[$l(n)$] *under linear metric reductions.*

- MINIMUM AXIOM SET [LO17]

  Instance: Finite weighted set of "sentences" S, subset $T \subseteq S$ of "true sentences," and "implication relation" R consisting of pairs (A, s) where $A \subseteq S$ and $s \in S$.

  Output: The minimum weight of any subset $S_0 \subseteq S$ such that the closure of S_0 under R includes T.

- CLUSTERING [MS9]

  Instance: Finite set X, integer distance $d(x, y)$ for $x, y \in X$ and integer k.

  Output: The minimum $\sum d(x, y)$ taken over pairs x, y in the same equivalence class for any partition of X into k classes.

- MAXIMUM PERMUTATION

  Instance: Integer n and polynomial time computable function f defined on permutations of $\{1, \ldots, n\}$.

  Output: The maximum value of $f(\sigma)$ taken over all permutations σ.

  Remark: See [Sah74].

- SQUARE TILING [GP13]

  Instance: Set C of colors, weighted set $T \subseteq C^4$ of tiles and integer N.

  Output: The maximum weight tiling of an $N \times N$ square where edges of adjacent tiles agree in color.

  Remark: In the unweighted version, tile weights are from $\{0, 1\}$. Both max and min versions are complete under exact metric reductions.

- UNIFICATION WITH COMMUTATIVE OPERATORS [AN16]

  Instance: Set V of variables, set C of constants, ordered pairs of expressions $(e_1, f_1), \ldots, (e_n, f_n)$, and weights $w_1, \ldots, w_n$ for the pairs.

  Output: The maximum weight of any assignment for the variables. An assignment associates a variable-free expression with each variable. The weight of an assignment is the sum of the w_i's such that e_i and f_i are equivalent under the substitution.

  Remark: An expression is a variable, a constant, or $(e + f)$ where e and f are expressions. Constants are equivalent if they are identical, and if e and e' are equivalent and f and f' are equivalent, then $(e + f)$, $(e' + f')$ and $(f' + e')$ are all equivalent. Also complete if we associate weights with the constants, define the weight of $(e + f)$ to be the sum of the weights on e and f, and ask for the minimum sum of weights of the expressions in the assignment that makes all of the pairs equivalent.

Proof: MINIMUM AXIOM SET. Straightforward from SET COVER.

Proof: CLUSTERING. Immediate from MAX CUT.

Proof: MAXIMUM PERMUTATION. Immediate from TRAVELING SALESMAN.

Proof: SQUARE TILING. Reduction from UNIV$[l(n)]$. Let M be a metric Turing machine computing UNIV$[l(n)]$, with tape alphabet Δ, states Q and running in time $p(n)$.

We may assume that M has one tape and that M makes at most k branches at any one step, for some k. Let $\Gamma = \Delta \cup \Delta \times Q \times \{1, \ldots, k\}$. An element in Γ represents either a tape symbol or a tape symbol together with a state and a non-deterministic choice (indicating that M's tape head scans this cell).

Let $w \in \Sigma^*$, let $n = |w|$ and let $N = p(n)$. We construct a set of tiles T such that any tiling of an $N \times N$ square represents a computation of $M(w)$. Each tile represents the contents of one tape cell (and possibly M's state), with the tape cells numbered $1, \ldots, N$ from left to right and time numbered $1, \ldots, N$ from bottom to top.

Note that the contents of tape cell i at time t depends only on cells $i - 1$, i and $i + 1$ at time $t - 1$. For $a, b, c \in \Gamma$, let $\delta(a, b, c) \subseteq \Gamma$ be the set of possible tape contents of cell i if a, b and c are the tape contents at cells $i - 1$, i and $i + 1$ at the previous step.

A color is either a 2-tuple from Γ and an integer (for the left and right sides) or a 3-tuple from Γ and an integer (for the top and bottom). The set of tiles (except for the first row) is: bottom $= (a, b, c, t)$, left $= (a', d, i)$, right $= (d, c', i + 1)$ and top $= (a', d, c', t + 1)$ for any $a, b, c, d, a', c' \in \Gamma$ and $0 \leq i, t \leq N - 1$, where $d \in \delta(a, b, c)$. The first row of tiles contains the initial tape contents and consists of the input w and the initial state, followed by blanks.

The weight of all of the tiles is 0 except for the tiles with time N. We may assume that M writes its output in binary on cells $1, \ldots, l(n)$ and erases the rest of its tape. Let a tile with top color (a, b, c, N) and right color (b, c, i) have weight $b2^{\,i}$, for $b \in \{0, 1\}$. Then the colors on the tiles represent the tape contents of a computation of $M(w)$ and the weight of the tiles is $\mathbf{opt}^M(w)$.

Proof: UNIFICATION. Reduction from 2-SAT. Let $\Phi(x_1, \ldots, x_n)$ be an instance of 2-SAT with clauses $C_1, \ldots, C_m$ and weights $w_1, \ldots, w_m$. Let $V = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n, y_1, \ldots, y_{2m}\}$ and let $C = \{0, 1\}$. For each variable x_i, make the pair $(x_i + \overline{x}_i)$ and $(0 + 1)$ with weight $M = \sum w_i$. For each clause $C_j = (a+b)$, make the pair $((a+b)+(y_{2j-1}+y_{2j}))$ and $((0 + 1) + (1 + 1))$ with weight w_j. An optimal assignment must satisfy all of the variable pairs and will have weight nM plus the maximum weight of Φ. With the weights on the constants, a similar reduction from LEX will work. Let $C = \{t_1, f_1, \ldots, t_n, f_n\}$, let the weight of t_i be 2^i and let the weight of f_i be 0. □

Theorem 3.13 *The following functions are complete for* OptP$[O(\log n)]$ *under linear metric reductions.*

- CODE GENERATION FOR TWO ADDRESS MACHINE [PO5]

  Instance: Expression dag (directed acyclic graph) G for a basic block of a program.

  Output: The length of the shortest sequence of instructions computing G on a two-address machine with unlimited registers.

  Remark: See [AJU77] or [GJ79] for more details.

- GEOMETRIC COVER

  Instance: Set S of points in the plane on integer coordinates and integer R.

  Output: The minimum number of circles of radius R centered on integer coordinates needed to cover all points in S.

  Remark: See [FPT81] or [BL89]. Also complete if we ask for the maximum number of points in S that can be covered by n circles of radius R. This last version with weights on the points in S is complete for OptP or OptP$[l(n)]$.

- MINIMUM INFERRED FINITE AUTOMATA (MIN DFA) [AL8]

  Instance: Finite alphabet Σ and finite subsets $A, R \subseteq \Sigma^*$.

  Output: The minimum number of states of a deterministic finite automata that accepts every string in A and rejects every string in R.

Proof: CODE GENERATION. The reduction from FEEDBACK VERTEX SET in [AJU77] is a linear reduction.

Proof: GEOMETRIC COVER. The construction in [FPT81] can be modified to give a reduction from CHEATING SAT.

Proof: MIN DFA. Reduction from COLORING. Let $G = (V, E)$ be a graph and let $V = \{0, \ldots, n-1\}$. Let $\Sigma = \{0, 1, x_0, \ldots, x_{n-1}\}$. Let A contain 0^{n+1}, $11, 011, \ldots, 0^{n-1}11$ and all strings $0^i 1 x_i$ for $0 \leq i \leq n-1$. Let R contain ε, 0, 00, $\ldots$, 0^n, 1, 01, $\ldots$, $0^n 1$ and all strings $0^i 1 x_j$ for every edge $(i, j) \in E$.

We identify a state in the DFA with a string that takes the initial state to it. The states for ε, 0, 00, $\ldots$, 0^{n+1} are all distinct because some number of 0's distinguishes them. The state for $0^i 1$ represents vertex i. These states are all distinct from any state 0^j because 1 distinguishes them. The minimum DFA accepting A and rejecting R consists of $n+2$ states for the strings ε, 0, $\ldots$, 0^{n+1} plus the states corresponding to the vertices. States $0^i 1$ and $0^j 1$ are compatible if and only if there is no edge $(i, j) \in E$ (otherwise x_i or x_j would distinguish them). Therefore, a minimum DFA corresponds to a legal coloring and the minimum number of states is $n+2$ plus the chromatic number of G. □

4 Open Problems

We conclude with a discussion of open problems. First, what is the precise number of NP queries needed to solve BIN PACKING? Approximating BIN PACKING has been a persistent open problem in approximation algorithms and it has a long history. Johnson et al. [JDU+74] showed an asymptotic ratio of 11/9. Fernandez de la Vega and Lueker [dlVL81] improved the ratio to $1 + \epsilon$ for any $\epsilon > 0$, and then Karmarkar and Karp [KK82] showed how to get within an additive constant of $O(\log^2 n)$. This last result implies that BIN PACKING is in $\mathrm{F}\Delta_2^p[2 \log\log n + O(1)]$ because we can do binary search on the interval it leaves.

Is there some function $l(n)$ such that computing the optimal number of bins is complete for $\mathrm{F}\Delta_2^p[l(n)]$? We know that BIN PACKING is hard for $\mathrm{F}\Delta_2^p[1]$ because it is NP-complete and that it can be solved with $O(\log\log n)$ queries, but its precise complexity is not known. Of course, it is possible that BIN PACKING cannot be solved with fewer than $O(\log\log n)$ queries but fails to be complete for this class. Alternatively, if we can't identify its exact complexity, maybe we can prove that BIN PACKING is equivalent to some other problems, perhaps OPTIMAL LINEAR ARRANGEMENT or BANDWIDTH. Also, it's still an open problem if BIN PACKING can be approximated using at most one bin more than optimal.

A second open question is the complexity of actually producing an optimal solution to an OptP problem and not just computing the optimal cost. For $\mathrm{OptP}[n^{O(1)}]$-complete problems such as TRAVELING SALESMAN, we can encode the solution in the low-order bits of the optimal cost. This implies that computing the optimal solution is equivalent to computing its cost for polynomially many queries. However, for fewer queries such as

$O(\log n)$, the situation is much less clear. For example, how do we find a maximum clique just by knowing its size? We know that if optimal solutions for even $\mathrm{F}\Delta_2^p[1]$-hard problems can be found with $l(n) \geq \log n$ queries, then $\mathrm{NP} \subseteq \textsc{Time}[2^{O(l(n))}]$ because we can try all possible sequences of oracle answers and then verify the solution. If $O(\log n)$ or fewer queries suffice, then P = NP. Again, it is possible that $O(n)$ queries are necessary but that it fails to be complete for this class.

Chen and Toda [CT91] have a partial solution for the case of $O(\log n)$ queries. Let $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{NP}}$ be the class of functions computable in polynomial time with a polynomial number of queries to an NP oracle but the queries must be made in parallel in one round. It is easy to see that $\mathrm{OptP}[O(\log n)]$ is contained in $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{NP}}$ because we can ask if the optimal value is greater than k for all possible k. Chen and Toda show that finding solutions to $\mathrm{OptP}[O(\log n)]$ problems can be randomly reduced to $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{NP}}$ and that for many natural $\mathrm{OptP}[O(\log n)]$-complete problems, finding solutions is $\mathrm{PF}_{\mathrm{tt}}^{\mathrm{NP}}$-hard.

Another question is what other subclasses of $\mathrm{F}\Delta_2^p$ and OptP have natural complete functions. We have already shown this for $\mathrm{F}\Delta_2^p[n^{O(1)}]$ and $\mathrm{F}\Delta_2^p[O(\log n)]$, and indeed the functional versions of most NP-complete problems are complete for one of these two classes. Of course, the characteristic function of any NP-complete problem is complete for $\mathrm{F}\Delta_2^p[1]$. What other parameters $l(n)$ have complete functions? For example, maybe BIN PACKING is complete for $O(\log \log n)$ queries.

Lastly, are there NP-complete problems whose functional versions are not OptP-complete? Our main result is that most NP-complete optimization problems are also $\mathrm{OptP}[l(n)]$-complete for some $l(n)$, but surely not every problem from Garey and Johnson [GJ79] will work. For example, some of the problems in algebra and number theory don't seem to fit well. QUADRATIC CONGRUENCES asks if there is an $x \leq c$ such that $x = a \pmod{b}$. We can easily make this an optimization function by asking for the largest such x, but the proof does not seem to work for this definition. If instead we ask for the largest (x mod d) satisfying the congruence, then we claim that the problem is OptP-complete. A similar result can be obtained for SIMULTANEOUS INCONGRUENCES. Also, some other problems such as BANDWIDTH and OPTIMAL LINEAR ARRANGEMENT have unusual optimization criteria and their proofs don't seem to generalize to OptP.

Acknowledgement

We thank Clyde Kruskal and Todd Wareham for proofreading the text and for pointing out an error in STEINER TREE.

References

[ABG90] Amihood Amir, Richard Beigel, and William I. Gasarch. Some connections between bounded query classes and non-uniform complexity. In *Proceedings of Fifth Annual IEEE Conference on Structure in Complexity Theory*, pages 232–243, 1990.

[AJU77] Alfred V. Aho, S. C. Johnson, and Jeffrey D. Ullman. Code generation for expressions with common subexpressions. *Journal of the ACM*, 24(1):146–160, January 1977.

[Ber85] Claude Berge. *Graphs*. Noth-Holland, Amsterdam, 1985.

[BL89] Chanderjit Bajaj and Ming Li. Geometric optimization and D^p-completeness. *Discrete and Computational Geometry*, 4:3–13, 1989.

[Coo71] Stephen Cook. The complexity of theorem-proving procedures. In *Proceedings of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[CT91] Zhi-Zhong Chen and Seinosuke Toda. On the complexity of computing optimal solutions. Manuscript, 1991.

[dlVL81] W. Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[FPT81] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, June 1981.

[Gas86] William I. Gasarch. The complexity of optimization functions. Technical Report 1652, Department of Computer Science, University of Maryland, 1986.

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

[Hol81] Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

[HU79] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[JDU+74] David S. Johnson, Alan Demers, Jeffrey Ullman, Michael R. Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[Kar72] Richard Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[KK82] N. Karmarkar and Richard Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 312–320, 1982.

[Kre88] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, June 1988.

[Lad75] Richard Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.

[LK78] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, January–February 1978.

[LY80] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20:219–230, 1980.

[Mai78] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, April 1978.

[Pap84] Christos H. Papadimitriou. On the complexity of unique solutions. *Journal of the ACM*, 31:392–400, 1984.

[PY84] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28:244–259, 1984.

[Sah74] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, December 1974.

[Ski85] Steven Skiena. Complexity of optimization problems on solitaire game Turing machines. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaigne, 1985.

[Ull75] Jeffrey D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[Wag88] Klaus W. Wagner. On restricting the access to an NP oracle. In *Proceedings of Fifteenth Annual International Colloquium on Automata, Languages and Programming*, pages 682–696. Springer-Verlag, Lecture Notes in Computer Science (number 317), 1988.

Figure 1: Tree of reductions.