On the Relative Complexity of 15 Problems Related to 0/1-Integer Programming

Andreas S. Schulz

Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, schulz@mit.edu

Summary. An integral part of combinatorial optimization and computational complexity consists of establishing relationships between different problems or different versions of the same problem. In this chapter, we bring together known and new, previously published and unpublished results, which establish that 15 problems related to optimizing a linear function over a 0/1-polytope are polynomial-time equivalent. This list of problems includes optimization and augmentation, testing optimality and primal separation, sensitivity analysis and inverse optimization, and several others.

1 Introduction

The equivalence of optimization and separation has been one of the most consequential results in combinatorial optimization, and beyond. For example, it gave rise to the first polynomial-time algorithms for finding maximum stable sets in perfect graphs, and for minimizing submodular functions, to mention just two of several results of this kind derived by Grötschel, Lovász, and Schrijver [9]. This equivalence has also paved the way for establishing negative results. For instance, it was instrumental in showing that computing the weighted fractional chromatic number is NP-hard [9]. For general linear optimization problems and corresponding polyhedra, the equivalence between optimization and separation holds under certain technical assumptions. For linear combinatorial optimization problems and associated 0/1-polytopes, however, these requirements are naturally satisfied. In fact, for such problems this equivalence not only means that a polynomial-time algorithm for one of the two problems implies that the other problem can be solved in polynomial time as well; if one of the two problems can be solved in strongly polynomial time, then so can the other [7, 10].

The relative computational complexity of solving one problem versus another has been studied in other situations as well. For example, Papadimitriou and Steiglitz wrote in their 1982 textbook on combinatorial optimization that "An Optimization Problem is Three Problems" [20, Page 343]. The other two

problems they referred to, in addition to finding an optimal solution, are computing the cost of an optimal solution—called the evaluation problem—and, in case of minimization problems, deciding whether there exists a feasible solution of cost at most a given value. The latter problem is known as the recognition or decision problem. It is obvious that the optimization problem is at least as hard as the evaluation problem, which, in turn, is no easier than the decision problem. Papadimitriou and Steiglitz went on by asking:

> Is it the case that all these versions are roughly of the same complexity? In other words, can we solve the evaluation version by making efficient use of a hypothetical algorithm that solves the recognition version, and can we do the same with the optimization and evaluation versions, respectively? [20, page 345]

Answers to these questions are known: Binary search over the range of possible values reduces the evaluation problem to solving a polynomial number of decision problems. The reduction from the optimization problem to the evaluation problem is oftentimes illustrated with the help of the Traveling Salesman Problem: Consider the arcs of the given directed graph in order, and for each arc solve the evaluation problem for the same instance where the cost of the current arc is increased by one. If the cost of an optimal tour is the same as before, then there is an optimal tour that does not use the current arc. The cost of this arc remains incremented, and the algorithm proceeds with the next arc. If, however, the cost of an optimal tour is higher than before, then the current arc is part of an optimal tour. Its cost is changed back to the original value, and the algorithm continues.

In this chapter, we prove that 15 problems, including augmentation, decision, evaluation, inverse optimization, optimization, primal separation, sensitivity analysis, separation, and testing optimality, are equivalent in Grötschel, Lovász, and Schrijver's and Papadimitriou and Steiglitz's sense: Given a hypothetical (strongly) polynomial-time algorithm for any one of them, each of the other problems can be solved in (strongly) polynomial time as well.

The chapter is organized as follows. In Section 2, we introduce the setup, in particular the class of linear combinatorial optimization problems considered here, and necessary background information, such as oracle-polynomial time algorithms. Section 3 constitutes the main part of this chapter. In Section 3.1, we present the 15 problems in detail and state the main result. Section 3.2 is reserved for its proof, which is broken down into several separate results pertaining to solving one problem with the help of a hypothetical algorithm for another problem. This part is followed by a collection of notes and references, which attribute the individual results to their respective authors (Section 3.3). Consequences and implications of the equivalence of the 15 problems are discussed in Section 4, including a proof of the Hirsch conjecture for 0/1-polytopes, a simplex-type algorithm for linear programming over 0/1-polytopes that visits only polynomially many vertices, and complexity results for exact local search. Finally, Sections 5 and 6 discuss extensions of

some of the results to local search and general integer linear programming, respectively.

2 Preliminaries

This chapter is concerned with linear combinatorial optimization problems. A linear combinatorial optimization problem Π consists of a family of instances $(N, \mathcal{F}, c)$, where $N = \{1, 2, \ldots, n\}$ is the ground set, $\mathcal{F} \subseteq 2^N$ is the set of feasible solutions, and the vector $c \in \mathbb{Z}^n$ assigns a cost of $\sum_{i \in F} c_i$ to every feasible solution $F \in \mathcal{F}$.[1] The objective is to find a feasible solution of minimal cost.[2] The set of feasible solutions of a combinatorial optimization problem is usually not described explicitly, but is given implicitly, such as the family of all stable sets in an undirected graph, or the set of all Hamiltonian tours in a directed graph. Typically, if we have one instance of such a combinatorial optimization problem, then the same instance in which only the objective function coefficients are changed, is also an instance of the same combinatorial optimization problem. We make this our first assumption.

Assumption 1. *Let Π be a linear combinatorial optimization problem. If $(N, \mathcal{F}, c)$ describes an instance of Π, and $d \in \mathbb{Z}^n$, then $(N, \mathcal{F}, d)$ is an instance of Π as well.*

Our interpretation of this assumption is that certain computational properties of the problems considered here, such as polynomial-time solvability, depend only on the structure of $\mathcal{F}$, but not on that of c.[3] As a consequence of Assumption 1, we henceforth use Π to just denote the family of pairs $(N, \mathcal{F})$, knowing that each such pair together with any compatible objective function vector constitutes an instance of the associated optimization problem. This convention allows us to refer to Π, even if we consider algorithmic problems other than optimization, which are defined over the instances $(N, \mathcal{F})$ of Π.

It is well known that the optimization problem associated with Π can be stated equivalently as a 0/1-integer programming problem. Namely,

$$\min\Big\{\sum_{i \in F} c_i : F \in \mathcal{F}\Big\} = \min\big\{cx : x \in X\big\} = \min\big\{cx : x \in P\big\},$$

where, for a given instance $(N, \mathcal{F}, c)$, $X := \big\{\chi^F : F \in \mathcal{F}\big\}$ is the set of incidence vectors of all feasible solutions, $P := \operatorname{conv}\{X\}$ is its convex hull, and cx

[1] The reason that these problems are called linear, or sometimes min-sum combinatorial optimization problems is that the cost functions are linear over the ground set.

[2] For the sake of definiteness, we consider minimization problems only. All results stated in this chapter apply equally to maximization problems.

[3] The most relevant exception to this premise are nonnegative cost coefficients, and we will comment later on which results of Section 3 remain valid if we restrict ourselves to such instances.

denotes the inner product of c and x, i.e., $cx = \sum_{i=1}^{n} c_i x_i$. The incidence vector χ^F of a feasible solution $F \in \mathcal{F}$ is defined as $\chi_i^F := 1$ if $i \in F$, and $\chi_i^F := 0$, otherwise. Thus, X offers an equivalent representation of feasible solutions as 0/1-points in the n-dimensional Euclidean space. Moreover, P is a 0/1-polytope, i.e., a polytope whose vertices have coordinates 0 or 1 only. Hereafter, we will use X to refer to the set of feasible solutions, as it will be convenient to work with 0/1-vectors instead of sets.

We assume that the reader is familiar with the concepts of polynomial-time and strongly polynomial-time algorithms, and we refer to [12] for more information on these and other notions of relevance to combinatorial optimization, which are not introduced here. One concept that is of particular importance to this chapter is that of an oracle-polynomial time algorithm, which helps us formalize the abstraction of a "hypothetical algorithm." An oracle-polynomial time algorithm is a polynomial-time algorithm that, in addition to all standard operations, may also make a polynomial number of calls to a given oracle. In particular, the time it would take to compute the answers that the oracle provides is not counted. However, if there were a polynomial-time algorithm for finding these answers, then one could replace the oracle by this algorithm, and the oracle-polynomial time algorithm would turn into a regular polynomial-time algorithm. The definition of a strongly oracle-polynomial time algorithm is similar. For technical details, we refer to [10, Chapter 1].

The concept of oracles and oracle-polynomial time algorithms is useful to relate the complexity of two different problems, defined over the same family Π of sets of feasible solutions, to one another. For instance, if there is an oracle-polynomial time algorithm for optimization that uses an oracle for the separation problem, then the optimization problem for Π is not harder than the separation problem for Π, at least as far as polynomial-time solvability is concerned. The concept of oracles also provides us with means to specify the way in which sets of feasible solutions are given computationally. For example, when we say that X is given by an evaluation oracle, an oracle-polynomial time algorithm for the optimization problem has no further knowledge of X than what it can acquire by calling the evaluation oracle a polynomial number of times.

We make one further assumption, which frees us from considering how to find an initial feasible solution, which often is a difficult problem in itself.

Assumption 2. *Let Π be a linear combinatorial optimization problem. For any instance (N, X) of Π, a feasible solution $x^0 \in X$ is explicitly known.*

Let us fix some additional notation. We use e_i to denote the i-th unit vector, which has a 1 in coordinate i and 0 elsewhere. Moreover, $\mathbb{1}$ is the all-one vector. Finally, the support of a vector $z \in \mathbb{Z}^n$ is defined by $\mathrm{supp}(z) := \{i \in N : z_i \neq 0\}$.

3 An Optimization Problem is 15 Problems

We start by detailing the computational problems defined over the instances (N, X) of a linear combinatorial optimization problem Π, which we will relate to one another.

3.1 List of Problems

Augmentation Problem (AUG). Given a feasible solution $x \in X$ and a vector $c \in \mathbb{Z}^n$, find a feasible solution $y \in X$ such that $cy < cx$, or state that x minimizes cx over X.

Component Determination Problem (CD). Given an index $i \in \{1, 2, \ldots, n\}$ and a vector $c \in \mathbb{Z}^n$, determine whether there exists an optimal solution x^* for minimizing cx over X such that $x_i^* = 1$.

Component Negation Problem (CN). Given an index $i \in \{1, 2, \ldots, n\}$, a vector $c \in \mathbb{Z}^n$, and an optimal solution x^* for minimizing cx over X, find an optimal solution for minimizing cx over $\{x \in X : x_i = 1 - x_i^*\}$, if one exists.

Decision Problem (DEC). Given a vector $c \in \mathbb{Z}^n$ and a number $K \in \mathbb{Z}$, decide whether X contains a feasible solution x with objective function value $cx \leq K$.

Evaluation Problem (EVA). Given a vector $c \in \mathbb{Z}^n$, find the cost of a solution minimizing cx over X.

Inverse Optimization Problem (INV). Given a feasible solution $x \in X$ and a vector $c \in \mathbb{Q}^n$, find a vector $d \in \mathbb{Q}^n$ such that x is optimal for minimizing dx over X and $\|c - d\|$ is minimal.[4]

Maximum Mean Augmentation Problem (MMA). Given a feasible solution $x \in X$ and a vector $c \in \mathbb{Z}^n$, find a feasible solution $y \in X$ such that $cy < cx$ and y maximizes $c(x - y)/|\mathrm{supp}(x - y)|$ over X, or state that x minimizes cx over X.

Optimization Problem (OPT). Given a vector $c \in \mathbb{Z}^n$, find a feasible solution that minimizes cx over X.

Postoptimality Problem (POST). Given an index $i \in \{1, 2, \ldots, n\}$, a vector $c \in \mathbb{Z}^n$, and an optimal solution x^* for minimizing cx over X, compute the maximal value $\rho_i \in \mathbb{Z}_+$ such that x^* remains optimal for minimizing $(c+\rho_i e_i)x$ and $(c - \rho_i e_i)x$ over X.

Primal Separation Problem (PSEP). Given a point $z \in [0, 1]^n$ and a feasible solution $y \in X$, find a vector $a \in \mathbb{Z}^n$ and a number $\beta \in \mathbb{Z}$ such that $ax \leq \beta$ for all $x \in X$, $ay = \beta$, and $az > \beta$, or state that such a vector a and number β do not exist.

[4] Instead of repeating the problem statement for different norms, let us merely remark that one can use the 1-norm or the ∞-norm. They would generally lead to different answers, of course. In terms of their computational complexity, however, they behave the same.

Separation Problem (SEP). Given a point $z \in [0,1]^n$, decide whether $z \in P = \text{conv}\{X\}$, and if not, find a vector $a \in \mathbb{Z}^n$ such that $ax < az$ for all $x \in P$.

Simultaneous Postoptimality Problem (SPOST). Given an index set $I \subseteq \{1, 2, \ldots, n\}$, a vector $c \in \mathbb{Z}^n$, and an optimal solution x^* for minimizing cx over X, compute the maximal value $\rho_I \in \mathbb{Z}_+$ such that x^* remains optimal for minimizing $(c + \sum_{i \in I} \delta_i e_i)x$ over X, for all $(\delta_i)_{i \in I}$ with $|\delta_i| \leq \rho_I$.

Unit Increment Problem (INC). Given an index $i \in \{1, 2, \ldots, n\}$, a vector $c \in \mathbb{Z}^n$, and an optimal solution x^* for minimizing cx over X, compute optimal solutions for the problems of minimizing $(c - e_i)x$ and $(c + e_i)x$ over X, respectively.

Verification Problem (VER). Given a feasible solution $x \in X$ and a vector $c \in \mathbb{Z}^n$, decide whether x minimizes cx over X.

Violation Problem (VIOL). Given a vector $a \in \mathbb{Z}^n$ and a number $\beta \in \mathbb{Z}$, decide whether $ax \geq \beta$ holds for all $x \in X$, and if not, find a feasible solution $y \in P = \text{conv}\{X\}$ with $ay < \beta$.

There are some obvious relationships between these problems. For instance, if we can solve the optimization problem in polynomial time, then we can also solve the augmentation problem and the evaluation problem in polynomial time. Moreover, a polynomial-time algorithm for evaluation gives rise to polynomial-time algorithms for the decision problem and the verification problem. And the polynomial-time solvability of augmentation or inverse optimization obviously implies the same for verification. Figure 1 shows all trivial relationships. Here, an arrow from one problem to another indicates that the problem at the head of the arrow can be solved in oracle-polynomial time given an oracle for the problem at the tail of the arrow. The relation is transitive, but arrows implied by transitivity are omitted.

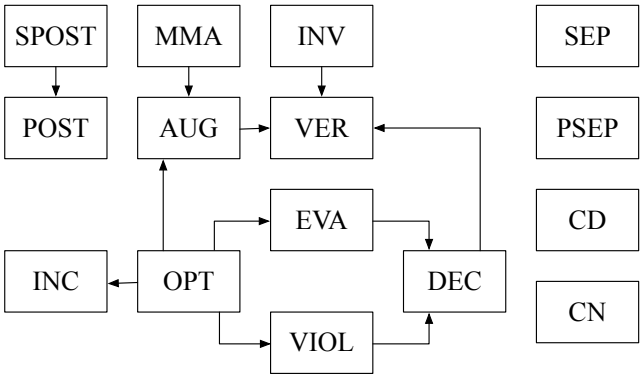

Fig. 1. Obvious relationships between the 15 problems

It turns out that, in fact, all these problems are equivalent, in terms of polynomial-time solvability.

Theorem 1. *Let Π be a linear combinatorial optimization problem. Any one of the following problems defined over the instances (N, X) of Π, augmentation, component determination, component negation, decision, evaluation, inverse optimization, maximum mean augmentation, optimization, postoptimality, primal separation, separation, simultaneous postoptimality, unit increment, verification, and violation, can be solved in oracle-polynomial time, given an oracle for any one of the other problems.*

This list is by no means exhaustive. While it contains some of the most interesting and relevant computational questions related to combinatorial optimization problems, other problems and variations of problems considered here can often proved to be equivalent by techniques similar to those illustrated in the following section.

3.2 Proof of Theorem 1

Instead of presenting a "minimal" proof of Theorem 1, it will be instructive to exhibit several direct reductions between different problems. In the end, there will be a "directed path," in the sense of Figure 1, from any one problem to any other problem, proving Theorem 1.

We first introduce some additional notation. For a given set $X \subseteq \{0,1\}^n$ of feasible solutions and an index $i \in \{1, 2, \ldots, n\}$, we define

$$X^{i,0} := \{x \in X : x_i = 0\} \text{ and } X^{i,1} := \{x \in X : x_i = 1\}.$$

Note that either $X^{i,0}$ or $X^{i,1}$ could be empty. For a given vector $c \in \mathbb{Z}^n$, and $l \in \{0,1\}$, let $x^{i,l}$ be an optimal solution for $\min\{cx : x \in X^{i,l}\}$, if one exists. The following observation will prove useful on several occasions.

Observation 2. *Let x^* be an optimal solution for minimizing cx over X.*

(a) *If $x^*_i = 1$, then $(c + \delta e_i)x^* \leq (c + \delta e_i)x$ for all $x \in X^{i,1}$ and $\delta \in \mathbb{Z}$.*
(b) *If $x^*_i = 0$, then $(c + \delta e_i)x^* \leq (c + \delta e_i)x$ for all $x \in X^{i,0}$ and $\delta \in \mathbb{Z}$.*

We start the proof of Theorem 1 by looking at sets of feasible solutions specified by an augmentation oracle. In addition to the trivial relationship depicted in Figure 1, we have the following three lemmata.

Lemma 3. *Assume that $X \subseteq \{0,1\}^n$ is given by an augmentation oracle. Then, the unit increment problem for X can be solved in oracle-polynomial time.*

Proof. Let the input to the unit increment problem be specified by x^*, c, and i. Let us first consider the case $x^*_i = 0$. Clearly, x^* stays optimal for

$c + e_i$. Hence, it suffices to focus on the objective function vector $c - e_i$. If x^* is no longer optimal for this vector, then $x^{i,1}$ is. As $(c - e_i)x^* - (c - e_i)x^{i,1} = cx^* - cx^{i,1} + 1 \leq 1$, one call to the augmentation oracle with input x^* and $c - e_i$ will suffice to obtain an optimal solution for $c - e_i$. The case $x_i^* = 1$ can be handled similarly. □

Lemma 4. *Assume that $X \subseteq \{0,1\}^n$ is given by an augmentation oracle. Then, there exists an oracle-polynomial time algorithm for the inverse optimization problem over X.*

Proof. Suppose we would like to solve the inverse optimization problem for x and c, and we are given access to an augmentation oracle. A first call to the oracle with input x and c will clarify whether x is optimal with respect to c. If it is, then we return c itself. Otherwise, we can formulate the inverse optimization problem for the 1-norm as the following linear program:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \lambda_i & & && \text{(1a)}\\
\text{subject to} \quad & \lambda_i \geq d_i - c_i & & \text{for } i = 1, 2, \ldots, n, && \text{(1b)}\\
& \lambda_i \geq c_i - d_i & & \text{for } i = 1, 2, \ldots, n, && \text{(1c)}\\
& dx \leq dy & & \text{for all } y \in X. && \text{(1d)}
\end{aligned}$$

Given a vector $(d, \lambda) \in \mathbb{Q}^n \times \mathbb{Q}^n$, the separation problem for the polyhedron defined by (1b)–(1d) can be solved as follows. Inequalities (1b) and (1c) can be checked directly. If one of them is violated, we obtain a separating hyperplane. If all of them are satisfied, we can call the augmentation oracle for x and d to find a violated inequality among (1d), if one exists. By the equivalence of separation and optimization, it follows that an optimal solution to the linear program (1) can be computed in polynomial time.

If we are dealing with the ∞-norm instead of the 1-norm, the argument remains essentially the same; we just replace the n variables λ_i by a single variable λ, and the objective function by λ. □

The next proof presents a reduction from optimization to augmentation, which, in contrast to the previous proof, does not rely on the equivalence of optimization and separation.

Lemma 5. *Assume that $X \subseteq \{0,1\}^n$ is given by an augmentation oracle. Then, the optimization problem for X can be solved in oracle-polynomial time.*

Proof. The proof is based on scaling. To simplify the exposition, we assume that the given objective function vector c has nonnegative entries. If this is not the case, we can switch x_i to $1 - x_i$ for all coordinates i with $c_i < 0$. The resulting objective function vector is nonnegative. Moreover, the augmentation oracle for the original set X of feasible solutions can be used to solve the augmentation problem for the new set of feasible solutions, after switching.

So, let $c \geq 0$. We may also assume that $c \neq 0$. Otherwise, any feasible solution is optimal.

Let $C := \max\{c_i : i = 1, 2, \ldots, n\}$, and let $K := \lfloor \log C \rfloor$. For $k = 0, 1, \ldots, K$, define c^k as the objective function vector with coefficients $c_i^k := \lfloor c_i / 2^{K-k} \rfloor$, $i = 1, 2, \ldots, n$. The algorithm works in phases. In phase k, it solves the optimization problem for c^k. A detailed description is provided below. As per Assumption 2, $x^0 \in X$ is a given feasible solution.

1: **for** $k = 0, 1, \ldots, K$ **do**
2: **while** x^k is not optimal for c^k **do**
3: $x^k := \text{AUG}(x^k, c^k)$
4: $x^{k+1} := x^k$
5: **return** x^{K+1}.

Here, $\text{AUG}(x^k, c^k)$ stands for the solution returned by the augmentation oracle when called with input x^k and c^k. The condition of whether to continue the while-loop is also checked during the same oracle call. The correctness of the algorithm follows from the fact that x^{K+1} is an optimal solution for $c^K = c$. Thus, it suffices to discuss the running time. There are only $\mathrm{O}(\log C)$, i.e., polynomially many phases. Consequently, the analysis comes down to bounding the number of times the augmentation oracle has to be called within any given phase k. Note that, at the end of phase $k-1$, x^k is optimal with respect to c^{k-1}, and hence for $2c^{k-1}$. Moreover, $c^k = 2c^{k-1} + c(k)$, for some 0/1-vector $c(k)$. Therefore, if x^{k+1} denotes the optimal solution for c^k at the end of phase k, we obtain

$$c^k(x^k - x^{k+1}) = 2c^{k-1}(x^k - x^{k+1}) + c(k)(x^k - x^{k+1}) \leq n.$$

The inequality is a consequence of the optimality of x^k for c^{k-1}, the fact that x^k and x^{k+1} are 0/1-points, and $c(k)$ being a 0/1-vector. As a result, the algorithm determines an optimal solution with at most $\mathrm{O}(n \log C)$ calls of the augmentation oracle. □

The proof of the following result is somewhat reminiscent of that of the "evaluation implies optimization" construction from the introduction. It is stated here to make the importance of "component determination" explicit.

Lemma 6. *Assume that $X \subseteq \{0,1\}^n$ is given by a component determination oracle. Then, the optimization problem for X can be solved in oracle-polynomial time.*

Proof. The following pseudo-code describes the reduction in detail. The condition in the if-construct of line 2 is equivalent to calling the component determination oracle.

1: **for** $i = 1$ to n **do**
2: **if** $x_i = 1$ for some optimal solution x with respect to c **then**
3: $c_i := c_i - 1$;

4: $y_i := 1$
5: **else**
6: $y_i := 0$
7: **return** y.

The algorithm maintains the following invariant:

> After iteration i, all optimal solutions for the current objective function vector are also optimal for the original objective function vector. Moreover, all solutions $x \in X$ that are optimal for the current objective function vector satisfy $x_1 = y_1, x_2 = y_2, \ldots, x_i = y_i$.

If this holds true, then, after n iterations, y constitutes an optimal solution. Obviously, this is true initially ($i = 0$). The proof is by induction over the number of iterations. Let us assume that the induction hypothesis is true for iteration i, and consider iteration $i + 1$.

If the oracle answers "no," then the $(i + 1)$-st coordinate of all optimal solutions for the current objective function vector is 0. While the algorithm records this by setting $y_{i+1} = 0$, the objective function vector remains unchanged. It follows by induction that the invariant is true after the conclusion of iteration $i + 1$.

If the oracle's reply is "yes," the change of the $(i+1)$-st objective function coefficient in line 3 renders all optimal solutions with $x_{i+1} = 0$ non-optimal. Optimal solutions with $x_{i+1} = 1$, of which there is at least one, remain optimal, however. The result follows. □

We now turn to the case in which we have access to a component negation oracle.

Lemma 7. *Assume that $X \subseteq \{0,1\}^n$ is given by a component negation oracle. Then, the unit increment problem for X can be solved in oracle-polynomial time.*

Proof. Let x^* and c be given, and suppose that $cx^* \leq cx$ for all $x \in X$. There are two cases. If $x^*_i = 1$, then x^* continues to be optimal for $c - e_i$. In order to determine an optimal solution for $c + e_i$, we call the component negation oracle with input x^*, c, and i. If there is no feasible solution $x \in X$ with $x_i = 0$, then x^* remains optimal. Otherwise, let $x^{i,0}$ be the solution returned by the oracle. Comparing the objective function values of x^* and $x^{i,0}$ with respect to $c + e_i$ yields the optimal solution. The case $x^*_i = 0$ is similar. □

Since we will later show that an algorithm for the optimization problem can be designed with the help of an oracle for the unit increment problem (Lemma 13), the following lemma is not needed, strictly speaking. However, the techniques involved are quite different and worth recording.

Lemma 8. *Let $X \subseteq \{0,1\}^n$ be given by a component negation oracle. Then, the optimization problem for X can be solved in oracle-poynomial time.*

Proof. Let x^0 be the given feasible solution. As usual, c is the objective function vector. The following algorithm determines an optimal solution x^*:

```
1: x* := x^0;
2: L := ∅;
3: d := c;
4: for i = 1, 2, ..., n do
5:    if (x*_i = 0 and c_i < 0) or (x*_i = 1 and c_i > 0) then
6:       L := L ∪ {i};
7:       d_i := 0
8: for all i ∈ L do
9:    call CN with input x*, d, and i;
10:   d_i := c_i;
11:   if CN returns a solution y* then
12:      if dy* < dx* then
13:         x* := y*
14: return x*.
```

The idea is the following. We first modify c such that x^0 becomes optimal (lines 4–7). Let d be the resulting vector. The algorithm maintains an optimal solution x^* for the modified objective function vector d, while incrementally changing it back to c. Initially, $x^* = x^0$. We then call the component negation oracle for each objective function coefficient d_i that differs from c_i. In each such iteration, the solution returned by the oracle, y^*, is optimal for d among all feasible solutions x with $x_i = y_i^* = 1 - x_i^*$. Moreover, x^* is optimal for d; in particular, x^* is optimal for d among all feasible solutions x with $x_i = x_i^*$. So when we change d_i to c_i (line 10), it follows that x^* or y^* is an optimal solution for minimizing dx over X. The algorithm makes the appropriate choice, and repeats the same argument with the next coefficient. □

The following proposition is not only quite useful in establishing relationships between the postoptimality problem and other problems; it also provides an interesting, if simple structural insight. Recall that ρ_i is the maximal value by which a coefficient c_i can be changed in either direction without causing a given optimal solution x^* to become non-optimal.

Proposition 9. *Let x^* be an optimal solution for minimizing cx over X. Let $i \in \{1, 2, \ldots, n\}$. If $x_i^* = 1$ and $X^{i,0} \neq \emptyset$, then $\rho_i = cx^{i,0} - cx^*$. If $x_i^* = 0$ and $X^{i,1} \neq \emptyset$, then $\rho_i = cx^{i,1} - cx^*$. Otherwise, $\rho_i = \infty$.*

Proof. Let us consider the case $x_i^* = 1$ first. Note that x^* remains optimal if we decrement c_i. Hence, ρ_i is only constrained by values that are greater than c_i. Clearly, x^* stays optimal for $c + \delta e_i$ if and only if $(c + \delta e_i)x^* \leq (c + \delta e_i)x$ for all $x \in X^{i,0}$. This is the case if and only if $\delta \leq cx - cx^*$ for all $x \in X^{i,0}$, which is equivalent to $\delta \leq cx^{i,0} - cx^*$, if $X^{i,0} \neq \emptyset$.

A similar argument yields $\rho_i = cx^{i,1} - cx^*$ when $x_i^* = 0$ and $X^{i,1} \neq \emptyset$. □

Proposition 9 renders the proof of our next result rather easy.

Lemma 10. *Let $X \subseteq \{0,1\}^n$ be given by a component negation oracle. Then, the postoptimality problem for X can be solved in oracle-polynomial time.*

Proof. Let x^*, c, and i be as specified in the description of the postoptimality problem. Use the component negation oracle to compute $x^{i,1-x_i^*}$, if it exists. If it does not exist, then $\rho_i = \infty$. Otherwise, $\rho_i = cx^{i,1-x_i^*} - cx^*$. □

The following result completes the formal proof of the "evaluation implies optimization" claim from the introduction; the second part, i.e., the step from component determination to optimization, is provided by Lemma 6.[5]

Lemma 11. *Assume that $X \subseteq \{0,1\}^n$ is given by an evaluation oracle. Then, the component determination problem for X can be solved in oracle-polynomial time.*

Proof. Solving the component determination problem with input $c \in \mathbb{Z}^n$ and $i \in \{1,2,\ldots,n\}$, requires just two calls of the evaluation oracle, one with the orginal vector c as input, and one with the modified vector $c - e_i$ as input. If the returned values are the same, the answer to the component determination problem is "no." Otherwise, it is "yes." □

We continue with another problem that can be solved with the help of an evaluation oracle.

Lemma 12. *The postoptimality problem for $X \subseteq \{0,1\}^n$ given by an evaluation oracle, can be solved in oracle-polynomial time.*

Proof. Let x^*, c, and i be the specified input of the postoptimality problem. Define $M := \sum_{k=1}^{n} |c_k| + 1$. We distinguish two cases. If $x_i^* = 1$, then we call the evaluation oracle with the objective function vector $c + Me_i$. Let $V(0)$ be the corresponding optimal value. Note that the i-th coordinate of any optimal solution with respect to $c + Me_i$ has to be 0, as long as there exists a feasible solution $x \in X$ with $x_i = 0$. Therefore, if $V(0) = cx^* + M$, then we return $\rho_i = \infty$. Otherwise, it follows from Proposition 9 that $\rho_i = V(0) - cx^*$.

If $x_i^* = 0$, then we feed the vector $c - Me_i$ into the evaluation oracle. Let $V(1)$ be the value returned by the oracle. We return $\rho_i = \infty$ if $V(1) = cx^*$, and $\rho_i = V(1) + M - cx^*$, otherwise. □

The algorithm establishing the next result differs only slightly from that presented in the proof of Lemma 5. Notwithstanding, we include its description here because it remains valid for general integer programs, as we will discuss in Section 6.

[5] The procedure sketched in the introduction implicitly uses a version of the component determination problem in which we query the existence of an optimal solution whose i-th component is 0, which can easily shown to be equivalent to the variant considered here.

Lemma 13. *Assume that $X \subseteq \{0,1\}^n$ is given by a unit increment oracle. Then, there exists an oracle-polynomial time algorithm that solves the optimization problem for X.*

Proof. As we did in the proof of Lemma 5, we may assume that the given objective function vector is nonnegative, applying the switching operation to coordinates with negative coefficients, if necessary. We also use the same notation as in the proof of Lemma 5. The algorithm proceeds in phases. Consider phase k, and let x^k be the optimal solution from the previous phase; i.e., $c^{k-1}x^k \leq c^{k-1}x$ for all $x \in X$. Recall that c^k arises from c^{k-1} as $c^k = 2c^{k-1} + c(k)$. In phase k, we consider all indices i with $c_i(k) = 1$ in order. Suppose $\{i : c_i(k) = 1\} = \{i_1, i_2, \ldots, i_l\}$. Then, $c^k = 2c^{k-1} + \sum_{h=1}^{l} e_{i_h}$. We therefore call the unit increment oracle in turn for the current optimal solution and $2c^{k-1} + e_{i_1}$, for the new optimal solution and $2c^{k-1} + e_{i_1} + e_{i_2}$, and so on. The optimal solution at the end of the phase is optimal for c^k, as required. □

We now make the connection to maximum mean augmentation.

Lemma 14. *Assume that $X \subseteq \{0,1\}^n$ is given by an optimization oracle. Then, there exists an oracle-polynomial time algorithm that solves the maximum mean augmentation problem for X.*

Proof. Let the input of the maximum mean augmentation problem be composed of x and c. A first call to the optimization oracle clarifies whether x minimizes cx over X. From now on, we assume it does not. We define $S := \mathrm{supp}(x)$, and we denote the value of the maximum ratio by μ^*. We are looking for a feasible point $y \in X$ such that $cy < cx$ and $c(x-y)/|\{i : x_i \neq y_i\}| = \mu^*$. Since x is not optimal, $\mu^* > 0$, and $cy < cx$ will be satisfied automatically if y is a feasible solution maximizing this ratio. Note that $\mu^* \leq C := \max\{|c_i| : i = 1, 2, \ldots, n\}$. We perform binary search over the interval $(0, C]$. For some value $0 < \mu \leq C$, we define an objective function vector c^μ as follows:

$$c_i^\mu := \begin{cases} c_i - \mu & \text{if } i \in S, \\ c_i + \mu & \text{if } i \notin S. \end{cases}$$

Suppose we call the optimization oracle with input c^μ; let x^μ be the output. There are three possible outcomes.

Case 1: $x^\mu = x$. Therefore, $c^\mu x \leq c^\mu z$ for all $z \in X$. Spelled out in detail, this means that $c(x-z) \leq \mu(|S| - \sum_{i \in S} z_i + \sum_{i \notin S} z_i)$, or, equivalently,

$$\frac{c(x-z)}{|\{i : x_i \neq z_i\}|} \leq \mu \text{ for all } z \in X \setminus \{x\}.$$

Accordingly, μ is an upper bound on μ^*.

Case 2: $x^\mu \neq x$ and $c^\mu x^\mu = c^\mu x$. This implies again that

$$\frac{c(x-z)}{|\{i : x_i \neq z_i\}|} \leq \mu \text{ for all } z \in X \setminus \{x\}.$$

However, x^μ satisfies this inequality with equality, and thus: $\mu = \mu^*$, and x^μ is the desired solution.

Case 3: $x^\mu \neq x$ and $c^\mu x^\mu < c^\mu x$. In this case,

$$\mu < \frac{c(x-x^\mu)}{|\{i : x_i \neq x_i^\mu\}|} \leq \mu^*.$$

Consequently, μ is a strict lower bound of μ^*.

Note that the absolute value of the difference of any two distinct ratios is at least $1/n^2$. Hence, after $O(\log(nC))$ steps and calls of the optimization oracle, binary search yields the optimal value μ^*. (There is a subtle technical detail: If the binary search procedure terminates with case 2, we get a corresponding solution automatically. Otherwise, the last lower bound resulted from case 1, and we do not explicitly have an optimal solution for c^μ that is different from the original point x. In this case, we perturb the objective function vector, so as to make sure that x is no longer optimal. An optimal solution x^μ different from x can then be attained by at most n additional oracle calls.) □

We follow up with another reduction relying on an optimization oracle.

Lemma 15. *Let $X \subseteq \{0,1\}^n$ be given by an optimization oracle. Then, the simultaneous postoptimality problem for X can be solved in oracle-polynomial time.*

Proof. In order to determine ρ_I, for $I \subseteq \{1,2,\ldots,n\}$, we need to find the largest value of ρ such that $\sum_{i=1}^n (c_i+\delta_i)x_i^* \leq \sum_{i=1}^n (c_i+\delta_i)x_i$ for all $x \in X$ and $\delta_i = 0$ for $i \notin I$, $|\delta_i| \leq \rho$ for $i \in I$. Equivalently, $\sum_{i\in I} \delta_i(x_i^* - x_i) \leq \sum_{i=1}^n c_i(x_i - x_i^*)$. Let us fix $x \in X$ and ρ for a moment. Then, the right-hand side of the last inequality is constant, and the inequality holds if and only if it holds for a vector δ with $|\delta_i| \leq \rho$ that maximizes the left-hand side. Consider some $i \in I$. If $x_i^* = 0$, then $\delta_i(x_i^* - x_i) = -\delta_i x_i$, which is maximized for $\delta_i = -\rho$, regardless of the value of x_i. We define $d_i := 1$. If $x_i^* = 1$, then $\delta_i(x_i^* - x_i) = \delta_i(1 - x_i)$, which is maximized for $\delta_i = \rho$, regardless of the value of x_i. We set $d_i := -1$. We also let $d_i := 0$ for $i \notin I$. Combining the pieces, we obtain that

$$\sum_{i\in I} \delta_i(x_i^* - x_i) \leq \sum_{i=1}^n c_i(x_i - x_i^*) \text{ iff } \rho \sum_{i\in I} x_i^* + \sum_{i=1}^n \rho d_i x_i \leq \sum_{i=1}^n c_i(x_i - x_i^*).$$

Thus, we need to find the largest ρ for which

$$\min_{x\in X} \sum_{i=1}^n (c_i - \rho d_i)x_i \geq \sum_{i=1}^n c_i x_i^* + \rho \sum_{i\in I} x_i^*. \tag{2}$$

Let us denote the left-hand side of (2) by $v(\rho)$. This function is piecewise linear and concave. The slope of $v(\rho)$ for any given ρ is equal to $-\sum_{i=1}^{n} d_i x_i$, for some $x \in X$. Since $d_i \in \{0, \pm 1\}$ and $x_i \in \{0, 1\}$, there can be at most $2n + 1$ different slopes. In other words, $v(\rho)$ has a small number of linear pieces. Before we discuss how to construct the relevant part of $v(\rho)$, let us quickly address over which interval we should do so.

Apparently, $\rho_I \geq 0$. If $x_i = x_i^*$ for all $i \in I$ and $x \in X$, then $\rho_I = \infty$. Note that one can easily check this condition by calling the optimization oracle with objective function vector $-\sum_{i \in I} (-1)^{x_i^*} e_i$. If x^* is optimal, then all $x \in X$ satisfy $x_i = x_i^*$, for all $i \in I$. Otherwise, ρ_I is finite, and $\rho_u := \sum_{i=1}^{n} |c_i| + 1$ is an upper bound on its value. From now on, we assume that ρ_I is finite.

We start by computing $v(\rho_u)$, which amounts to calling the optimization oracle once. If (2) is satisfied for $\rho = \rho_u$, we are done. Otherwise, the optimal solution associated with $v(\rho_u)$ defines a linear function on $[0, \rho_u]$. We compute the largest value of ρ for which the value of this linear function is greater than or equal to the right-hand side of (2). We denote this value by ρ_1 and compute $v(\rho_1)$. If $v(\rho_1)$ coincides with the value of the linear function at the point ρ_1, then $\rho_I = \rho_1$. Otherwise, the optimal solution corresponding to $v(\rho_1)$ defines a second linear function. We then compute the largest value of ρ for which this linear function is greater than or equal to the right-hand side of (2); let this value be ρ_2. Note that $\rho_2 < \rho_1$. We repeat the same procedure for ρ_2, and so on. Because $v(\rho)$ has $O(n)$ linear pieces, ρ_I is found after as many steps. □

The following two results are of a more polyhedral nature. Let us establish first that primal separation is no harder than ordinary separation.

Lemma 16. *Assume that $X \subseteq \{0, 1\}^n$ is given by a separation oracle. Then, the primal separation problem for X can be solved in oracle-polynomial time.*

Proof. Let $z \in [0, 1]^n$ be the given point, and let $y \in X$ be the given feasible solution, which together form the input of the primal separation problem. Without loss of generality, we may assume that $y = 0$. Let C be the polyhedral cone that is defined by the inequalities of $P := \text{conv}\{X\}$ that are satisfied with equality by y. The key observation is that the primal separation problem for P and y is essentially equivalent to the separation problem for C. The only difference is that a hyperplane $ax = \beta$ that separates z from C may not contain y. However, this can be fixed by pushing the hyperplane towards C until they touch. Put differently, $ax = 0$ is a separating hyperplane, too.

As we are dealt a separation oracle for P, and not for C, it remains to show how we can use the given oracle to emulate one for C. Figure 2 provides a schematic picture of the situation. If we called the separation oracle for the original point z, it could happen that $z \in C \setminus P$, and the oracle would return a separating hyperplane when, in fact, $z \in C$. By choosing a point $\bar{z}$ on the line between y and z, close enough to y, this can be prevented. We now prove that $\bar{z} := (1 - \frac{1}{n})y + \frac{1}{n}z \in P$, if $z \in C$. Let $y^1, y^2, \ldots, y^k$ be the vertices of P that are adjacent to y. The distance of y, measured in terms of the 1-norm, to

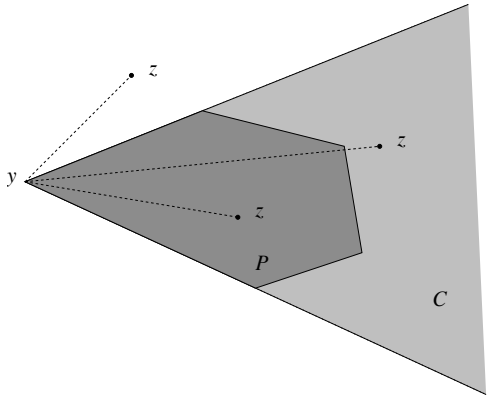

Fig. 2. Schematic drawing of P, C, and potential locations of z relative to y

any of its adjacent vertices is at least 1. This implies that the distance of y to any point in C that is not a convex combination of $y, y^1, y^2, \ldots, y^k$, is strictly greater than one. However, $z \leq \mathbb{1}$ implies $\|\bar{z}\|_1 \leq 1$. Therefore, $\bar{z}$ is contained in P, as we wanted to show.

At last, observe that when z does not belong to C, $\bar{z}$ does not belong to C, either. This completes the proof. □

The next proof relies on the equivalence of optimization and separation.

Lemma 17. *Let $X \subseteq \{0,1\}^n$ be given by a primal separation oracle. Then, the verification problem for X can be solved in oracle-polynomial time.*

Proof. Recall that verification means deciding whether a given feasible solution $y \in X$ is optimal with respect to a given objective function vector $c \in \mathbb{Z}^n$. Let $P := \text{conv}\{X\}$, and let C be the polyhedral cone defined by the linear inequalities defining P that are active at y. It follows from linear programming duality that y minimizes cy over P if and only if it minimizes cy over C. By the equivalence of optimization and separation, minimizing cy over C is equivalent to solving the separation problem for C. As explained in the proof of Lemma 16, the separation problem for C can be reduced to the primal separation problem for P and y. Hence, we can use the primal separation oracle to optimize over C. If the resulting value is finite, then y is an optimal solution. Otherwise, y is not optimal. □

We proceed with the postoptimality oracle, and discuss how it can be used to solve the component negation problem.

Lemma 18. *The component negation problem for $X \subseteq \{0,1\}^n$ given by a postoptimality oracle, can be solved in oracle-polynomial time.*

Proof. The input of the component negation problem is specified by an objective function vector c, an optimal solution x^*, and an index i. Proposition 9

implies that a single call to the postoptimality oracle suffices to compute the optimal value of the problem of minimizing cx over $\{x \in X : x_i = 1 - x_i^*\}$. However, we are interested in determining a corresponding optimal solution, not just its value. Of course, if the postoptimality oracle outputs $+\infty$, then there exists no solution $x \in X$ with $x_i = 1 - x_i^*$, and we can stop. Otherwise, the algorithm below describes how this can be done. We use CN-VAL(x^*, c, i) to denote the optimal value of $\min\{cx : x \in X, x_i = 1 - x_i^*\}$. For the sake of simplifying the exposition, we assume that $x^* = 0$. Similarly to the proof of Lemma 5, this can be achieved by switching coordinates of value 1 to 0, and by observing that the postoptimality oracle for the original problem can be used to solve the postoptimality problem for the transformed problem.

1: $V :=$ CN-VAL(x^*, c, i);
2: $y_i^* := 1$;
3: **for all** $k \in \{1, 2, \ldots, n\} \setminus \{i\}$ **do**
4: $\quad c_k := c_k + 1$;
5: $\quad$ **if** CN-VAL$(x^*, c, i) > V$ **then**
6: $\quad\quad y_k^* := 1$;
7: $\quad\quad c_k := c_k - 1$
8: $\quad$ **else**
9: $\quad\quad y_k^* := 0$
10: **return** y^*.

Basically, this algorithm imitates the evaluation-to-optimization procedure from the introduction, except that it has to use the postoptimality oracle instead of the evaluation oracle. In particular, we need to ensure that x^* remains an optimal solution for the current objective function vector c at all times. Otherwise, we would not be able to make calls to the postoptimality oracle in line 5. However, this is obvious because we are only raising c_k if $x_k^* = 0$. The rest of the proof is similar to that of showing that a polynomial-time algorithm for evaluation can be used to solve the optimization problem in polynomial time as well. □

Finally, we turn to two problems that can be directly solved by a verification oracle.

Lemma 19. *The augmentation problem for $X \subseteq \{0,1\}^n$ given by a verification oracle, can be solved in oracle-polynomial time.*

Proof. Let x and c specify the input to the augmentation problem. We may assume that $x = \mathbb{1}$. Indeed, if $x \neq \mathbb{1}$, we can replace all coordinates of the form $x_i = 0$ by $\bar{x}_i := 1 - x_i$. Formally, applying this switching operation transforms the given set of feasible solutions, X, into another set of feasible solutions, $\bar{X}$. By switching the signs of the corresponding entries c_i in the objective function, it is easy to see that x is optimal with respect to c if and only if $\mathbb{1}$ is optimal for the new objective function vector. The same argument shows that a verification oracle for X yields one for $\bar{X}$.

A first call of the verification oracle with $x = \mathbb{1}$ and c as input will determine whether x is optimal with respect to c. If it is not, we identify a better feasible solution with the help of the following algorithm.

1: $M := \sum_{i=1}^{n} |c_i| + 1$;
2: **for** $i = 1$ to n **do**
3: $\quad c_i := c_i - M$;
4: $\quad$ call the verification oracle with input x and c;
5: $\quad$ **if** x is optimal **then**
6: $\quad\quad y_i := 0$;
7: $\quad\quad c_i := c_i + M$
8: $\quad$ **else**
9: $\quad\quad y_i := 1$
10: **return** y.

Because c may change during the course of the algorithm, we introduce the following terminology to simplify the subsequent exposition. We say that a feasible solution is "better" than x if the objective function value of that solution with respect to the original vector c is smaller than that of x. Moreover, we will use c^i to denote the objective function defined by line 3 in iteration i of the algorithm.

It is instructive to go carefully over the first iteration, not only because it will serve as the base case for the inductive proof, but also because it lays out the kind of arguments that we will use in the inductive step. For $i = 1$, the verification oracle is called with an objective function vector whose first coordinate has been changed to $c_1 - M$. If x is optimal for c^1, we claim that the first component of *all* feasible solutions that are better than x has to be 0. Suppose not. Then there exists a solution $z \in X$ with $cz < cx$ and $z_1 = 1$. But then, $c^1 z = cz - M < cx - M = c^1 x$, which is a contradiction. We record this by setting $y_1 = 0$ in line 6 of the algorithm. On the other hand, we claim that, if x is not optimal for c^1, then there must exist *some* better feasible solution whose first component is 1. Again, suppose this was not the case. Let $z \in X$ be an arbitrary solution that is better than x; i.e., $cz < cx$. According to our assumption, $z_1 = 0$. Note that $cx - cz \leq \sum_{i=1}^{n} |c_i| < M$. Therefore, $c^1 x = cx - M < cz = c^1 z$, a contradiction. Hence, there has to be a better solution whose first coordinate is 1. We keep this in mind by setting $y_1 = 1$ in line 9. Moreover, in this case we do *not* revert c_1 to its original value. Keeping $c_1 - M$ ensures that, from now on, we are only looking for solutions better than x whose first component is 1.

We now continue by induction over the number of iterations. Our induction hypothesis is the following:

> After iteration i, there exists a better feasible solution whose first i components are equal to $y_1, y_2, \ldots, y_i$, respectively.

It follows from the discussion above that the induction hypothesis is satisfied after the first iteration. So let us assume that it is true after i iterations, and

consider iteration $i+1$. Note that

$$c^{i+1} = c - \sum_{k=1}^{i} y_k M e_k - M e_{i+1}.$$

Let us first consider the case that x is optimal for c^{i+1}. Suppose there exists a better solution $z \in X$ with $z_k = y_k$ for $k = 1, 2, \ldots, i$, and $z_{i+1} = 1$. Then,

$$c^{i+1} z = cz - M \left(\sum_{k=1}^{i} y_k + 1 \right) < cx - M \left(\sum_{k=1}^{i} y_k + 1 \right) = c^{i+1} x.$$

Hence, all solutions z with $z_k = y_k$ for $k = 1, 2, \ldots, i$, which are better than x, have $z_{i+1} = 0$. If x is not optimal with respect to c^{i+1}, let z be a better solution such that $z_k = y_k$ for $k = 1, 2, \ldots, i$, and $z_{i+1} = 0$. Since $cx < cz + M$, we obtain

$$c^{i+1} x = cx - M \left(\sum_{k=1}^{i} y_k + 1 \right) < cz - M \left(\sum_{k=1}^{i} y_k + 1 \right) + M = c^{i+1} z.$$

Consequently, there exists a better solution z such that $z_k = y_k$ for $k = 1, 2, \ldots, i$, and $z_{i+1} = 1$.

It follows that, after n iterations, y is a better feasible solution than x. □

The next result is much simpler to obtain.

Lemma 20. *Assume that $X \subseteq \{0,1\}^n$ is given by a verification oracle. Then, the postoptimality problem for X can be solved in polynomial time.*

Proof. An input of the postoptimality problem is specified by a vector c, an optimal solution x^*, and an index i. We discuss here the case that $x^*_1 = 1$. The other case, $x^*_i = 0$, is similar. Let $M := \sum_{k=1}^{n} |c_k| + 1$, and $C := \max\{|c_k| : k = 1, 2, \ldots, n\}$. We perform binary search over the interval $[0, M]$ to identify the largest value of δ such that x^* remains optimal for $c + \delta e_i$. As all data are integral, this requires $O(\log(nC))$ calls of the verification oracle. Moreover, if x^* is optimal for $c + Me_i$, then x^* continues to be optimal for all values of δ; i.e., $\rho_i = \infty$. This completes the proof. □

Actually, this completes the proof not only of Lemma 20, but also of Theorem 1. We refer to [10, Theorem 6.4.9] for a proof of the missing equivalence between optimization, separation, and violation.

3.3 Notes and References

All primal algorithms[6] for solving specific combinatorial optimization problems, i.e., algorithms that move from one feasible solution to the next while

[6] Primal algorithms are sometimes also referred to as exact local search algorithms, see Section 4.2 below.

improving the objective function value, solve, by definition, the augmentation problem in each iteration. This includes well-known algorithms such as cycle-canceling algorithms for min-cost flow problems and augmenting path algorithms for matching problems. However, the design of these algorithms often seems to require finding particular augmenting structures, such as cycles of minimum mean cost, in order to guarantee that the total number of iterations is polynomial in the input size. The universal applicability of the bit-scaling framework in the proof of Lemma 5 was not realized before 1995, when Grötschel and Lovász [8] and Schulz, Weismantel, and Ziegler [25] first published this result. Grötschel and Lovász underlined its usefulness by devising a simple polynomial-time algorithm for finding a maximum-weight Eulerian subdigraph in an arc-weighted directed graph. The paper by Schulz, Weismantel, and Ziegler contained some other equivalences, including that between optimization and "irreducible" augmentation, a concept relevant to test sets in integer programming [31], and that of optimization and maximum mean augmentation (Lemma 14), parts of which were later extended to general integer programming; see Section 6 below.

The verification and the component determination problems both go back to a paper by Papadimitriou and Steiglitz [19], in which they showed that the following two problems are NP-hard for the Traveling Salesman Problem:

> Given an instance and a tour x, is x suboptimal?
>
> Given an instance and an edge e, does e not appear in any optimal tour?

Lemmata 6 and 19, first brought up in [26], imply that these two problems are in fact NP-hard for every NP-hard linear combinatorial optimization problem.

As mentioned in the introduction, the equivalence of decision, evaluation, and optimization is widely known, but it appears difficult to pin down its original source. The direction from evaluation to optimization has sometimes been proved using the self-reducibility that certain problems, such as Maximum Satisfiability, exhibit [29, Chapter A.5]. Exploiting this property effectively amounts to the explicit fixing of variables, which can equally be achieved by modifying objective function coefficients accordingly, without necessitating a change of the set X of feasible solutions. If we look beyond the realm of linear combinatorial optimization problems, the question whether evaluation is as hard as optimization has been settled in the affirmative for all optimization problems whose associated decision problems are NP-complete [2, Chapter 1.4.4]. In related work, Crescenzi and Silvestri [4] provided sufficient and necessary conditions for the existence of optimization problems for which obtaining an optimal solution is harder than computing the optimal cost.

The unit increment problem, which together with Lemmata 3 and 13 is put forward in [16], is included here because Lemma 13 remains valid for general

integer linear programming problems. Section 6 has more on this topic as well as on the equivalence of evaluation and optimization in this broader context.

Ahuja and Orlin [1] showed that if a linear combinatorial optimization problem can be solved in polynomial time, then so can its inverse problem. Lemma 4 sharpens their result a bit, at the same time establishing a direct connection between augmentation and separation, which was speculated on earlier in [26].

The origins of primal separation trace back to primal cutting plane algorithms for integer programming; see, e.g., [18, 33] for earlier studies, and [13] for a more recent account. Padberg and Grötschel [17, Exercise 7] noted that the primal separation problem can be transformed into the standard separation problem; the proof of Lemma 16 uses the particularities of 0/1-polytopes to simplify the argument. The reverse direction, especially that primal separation implies augmentation, was established by Eisenbrand, Rinaldi, and Ventura [5]. Their original proof is based on solving a constrained version of the verification problem. In this variant, an index set $I \subseteq \{1, 2, \ldots, n\}$ is added to the input, and the question is whether x minimizes cy over $\{y \in X : y_i = x_i \text{ for all } i \in I\}$. However, it is not difficult to see that this version is polynomial-time equivalent to the ordinary verification problem. In fact, this is true for similar variants of other problems, such as component determination.

The component negation problem was introduced by Ramaswamy and Chakravarti [21] in their proof of the equivalence of optimization and postoptimality. The proof presented here is a streamlined version of the original proof; in particular, Lemma 8, Proposition 9, Lemma 10, Lemma 12, and Lemma 18 are all contained, either explicitly or implicitly, in their paper. Chakravarti and Wagelmans [3] generalized this result to allow for simultaneous changes of more than one objective function coefficient. In particular, they proved Lemma 15 and also showed that a similar approach works if the deviation from the original objective function coefficients is measured in a relative sense instead of in terms of absolute differences. They also addressed situations in which the objective function coefficients are restricted to be nonnegative, as was previously done for the single variable postoptimality problem in [21, 28].

More generally, a second look at the proofs in Section 3.2 reveals that most results remain valid if the objective function coefficients are restricted to be nonnegative for both the given oracle and the problem to be solved. Yet, some problem statements have to be adjusted accordingly. For instance, if we do not restrict the objective function coefficients in any way, it is straightforward to show that the version of the component determination problem considered here is polynomial-time equivalent to the one in which we inquire about the existence of an optimal solution whose i-th coordinate is 0. For nonnegative cost coefficients, however, this is not true. While the latter version remains equivalent with optimization and evaluation, it is NP-complete to decide, for example, whether a given directed graph with nonnegative arc costs has a shortest s-t-path that contains a specific arc [6].

3.4 Weakly vs. Strongly Polynomial Time

Some of the reductions in the proof of Theorem 1 lead to weakly polynomial time algorithms, and one may wonder whether the 15 problems are strongly polynomial time equivalent. As far as linear combinatorial optimization problems are concerned, Frank and Tardos [7] provided a rounding scheme that resolves this matter for good:

Lemma 21. *There exists an algorithm that, for a given positive integer $Q \in \mathbb{Z}_+$ and a rational vector $c \in \mathbb{Q}^n$, finds an integer vector $d \in \mathbb{Z}^n$ such that $\|d\|_\infty \leq 2^{4n^3} Q^{n(n+2)}$ and $\operatorname{sign}(cz) = \operatorname{sign}(dz)$ whenever z is an integer vector with $\|z\|_1 \leq Q-1$. Moreover, the running time of this algorithm is polynomial in n and $\log Q$.*

For example, with $Q := n+1$, given an objective function vector $c \in \mathbb{Q}^n$, we obtain another objective function vector $d \in \mathbb{Z}^n$ such that $cx \leq cy$ if and only if $dx \leq dy$, for all $x, y \in \{0,1\}^n$. Moreover, the size of d is polynomial in n: $\log \|d\|_\infty = \mathrm{O}(n^3)$. For instance, if we apply this rounding procedure prior to the bit-scaling algorithm described in the proof of Lemma 5, then it follows that the algorithm described therein runs in strongly oracle-polynomial time. Similar preprocessing is possible in other cases, yielding:

Corollary 22. *Let Π be a linear combinatorial optimization problem. If any one of the 15 problems listed in Theorem 1 can be solved in strongly polynomial time, then so can any other problem from this list.*

4 Further Consequences

There are two ways of looking at Theorem 1. On the one hand, it is a tool to prove negative results. If a linear combinatorial optimization problem is NP-hard, then so are several other computational aspects related to that problem. Sensitivity analysis is hard, finding another optimal solution for a nearby objective function is hard, and so is recognizing an optimal solution. On the other hand, if the problem can be solved in polynomial time, it is convenient to know, for instance, that the simultaneous postoptimality problem can be solved in polynomial time as well. Moreover, Theorem 1 provides a variety of means to show that a linear combinatorial optimization problem can be solved in polynomial time, giving the algorithm designer some choice. Depending on the circumstances, one way of attack may prove to be simpler than another. For example, solving the primal separation problem of the Perfect Matching Problem in some graph $G = (V, E)$ requires only $|V|/2$ elementary maximum flow computations [5], compared to more complicated primal augmentation algorithms or minimum odd cut algorithms for solving the ordinary separation problem.

We continue by exploring some other implications of Theorem 1.

4.1 0/1-Polytopes and Linear Programming

As discussed earlier, optimizing cx over $X \subseteq \{0,1\}^n$ is equivalent to optimizing cx over $P := \text{conv}\{X\}$. By definition, P is a 0/1-polytope; i.e., all its vertices have coordinates 0 or 1. Two vertices of P are adjacent if they belong to the same edge of P. Thanks to the following well-known fact from linear programming, Lemma 5 has implications for the combinatorial structure of 0/1-polytopes, and for linear programming over 0/1-polytopes, which are thus far elusive for general polytopes.

Fact 23. *Let P be a polytope, and consider $\min\{cx : x \in P\}$, for some $c \in \mathbb{Z}^n$. If z is a non-optimal vertex of P, then there exists an adjacent vertex $y \in P$ such that $cy < cz$.*

The vertices and edges of a polytope P form an undirected graph. The diameter of P is the diameter of that graph; that is, it is the smallest number δ such that any two vertices of P are connected by a path consisting of at most δ edges. Lemma 5 yields an alternate proof of the following result, due to Naddef [14].

Lemma 24. *The diameter of a 0/1-polytope is at most its dimension.*

Proof. Let $P \subseteq [0,1]^n$ be a 0/1-polytope. Without loss of generality, we may assume that P is full-dimensional (see, e.g., [34, Chapter 3.3]). Let y and z be two distinct vertices of P. If we define the vector c by setting

$$c_i := \begin{cases} 1 & \text{if } y_i = 0, \\ -1 & \text{if } y_i = 1, \end{cases} \qquad \text{for } i = 1, 2, \ldots, n,$$

then y is the unique minimum of cx over P. According to the proof of Lemma 5, starting from z, we obtain y after at most n calls of the augmentation oracle. By Fact 23, we may assume that the oracle, when fed with a non-optimal vertex x, outputs a vertex that is adjacent to x. The vertices returned by the oracle form the required path of length at most n. □

Naddef [14] went on to show that Lemma 24 implies that the Hirsch conjecture is true for 0/1-polytopes:

Corollary 25. *The diameter of a 0/1-polytope is bounded by the number of its facets minus its dimension.*

Proof. We include a proof for the sake of completeness. Let P be a 0/1-polytope, and let f be its number of facets, and n its dimension. If $f \geq 2n$, we are done, by Lemma 24. If $f < 2n$, any two vertices of P lie on a common facet, and the result follows by induction over the dimension. □

It is well known that the diameter of a polytope is a lower bound for the number of iterations needed by the simplex algorithm, regardless of the pivot rule used. The following result provides a corresponding upper bound, for a particular variant of the simplex algorithm.

Theorem 26. *Let $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$, and $x^0 \in \{0,1\}^n$ with $Ax^0 \geq b$ be given. Moreover, assume that $P := \{x \in \mathbb{R}^n : Ax \geq b\}$ is a 0/1-polytope. Then, there exists a variant of the simplex algorithm that solves $\min\{cx : x \in P\}$ by visiting at most $O(n^4)$ vertices of P.*

Proof. In a first step, we use the polynomial-time algorithm of Lemma 21 with input c and $Q := n + 1$ to replace the given objective function vector c by another objective function vector $d \in \mathbb{Z}^n$. As a result, $\log \|d\|_\infty = \mathrm{O}(n^3)$, and, for any two points $x, y \in \{0,1\}^n$, $cx \leq cy$ if and only if $dx \leq dy$. According to the algorithm given in the proof of Lemma 5, we then proceed in phases. In phase k, we minimize the objective function vector obtained by only considering the k most significant bits of every coefficient of d. Each phase starts with the optimal solution from the previous phase, with the exception of the first phase, which starts with x^0. In each phase, we use the simplex algorithm with any monotone pivot rule to find an optimum. As all vertices of P are 0/1-points, Lemma 5 implies the result. □

We conclude this section by pointing out that the path of vertices traced by the algorithm described in the proof of Theorem 26 is not necessarily monotone in d, nor in c.

4.2 Exact Local Search

The simplex algorithm has often been rightly characterized as a local search algorithm, where the neighborhood of a vertex contains its adjacent vertices (e.g., [20]). By Fact 23, this neighborhood is exact.[7] Conversely, the unique minimal exact neighborhood of a linear combinatorial optimization problem Π is precisely the one that would be searched by the simplex algorithm on $P = \mathrm{conv}\{X\}$ [23]. It turns out that Theorem 1 subsumes some generalizations of problem-specific results pertaining to exact neighborhoods.

After Savage, Weiner and Bagchi [22] had shown that any exact neighborhood for the Traveling Salesman Problem has to have exponential size, Papadimitriou and Steiglitz [19] formulated the following result for the TSP.

Corollary 27. *If P≠NP, no NP-hard linear combinatorial optimization problem can have an exact neighborhood that can be searched in polynomial time for an improving solution.*

Proof. Let Π be an NP-hard linear combinatorial optimization problem, and let N be an exact neighborhood. If one can check in polynomial time whether the current feasible solution is locally (= globally) optimal, and, if not, produce a better feasible solution, then there is a polynomial-time algorithm for

[7] In general, a neighborhood function N for a linear combinatorial optimization problem Π is called exact, if, for any instance X, any locally optimal solution is already globally optimal; i.e., $cx \leq cy$ for all $y \in N(x)$ implies $cx \leq cy$ for all $y \in X$, for all $c \in \mathbb{Z}^n$.

solving the augmentation problem. Hence, by Lemma 5, Π can be solved in polynomial time, which would imply P=NP. □

For linear combinatorial optimization problems, Corollary 27 refines earlier results of Tovey [27] and Yannakakis [32, Theorem 18], which needed the stronger assumption of either NP≠co-NP, or that the problem be strongly NP-hard.

5 Local vs. Global Optimality

In light of the results for exact neighborhoods, it is natural to ask whether some of them remain valid in situations where local optima are not necessarily globally optimal. In particular, it often is the case that a neighborhood can be searched fast, either because it is of polynomial size (examples include the k-exchange neighborhood for the Traveling Salesman Problem for some constant k, the flip neighborhood for the maximum Cut Problem, and the swap neighborhood for the Graph Partitioning Problem), or because there is an efficient algorithm to identify an improving solution in the neighborhood, if one exists (examples of neighborhoods of exponential size that can be searched in polynomial time include the twisted sequences neighborhood, the pyramidal tours neighborhood, and the permutation tree neighborhood for the Traveling Salesman Problem). Put differently, given a linear combinatorial optimization problem Π together with a neighborhood N, the "local" version of the augmentation problem can typically be solved in polynomial time.[8]

Local Augmentation Problem. Given a feasible solution $x \in X$ and a vector $c \in \mathbb{Z}^n$, find a feasible solution $y \in N(x)$ such that $cy < cx$, or state that x is locally optimal with respect to N and c.

However, an analogous result to Lemma 5 is not known. That is, given a local augmentation oracle, we do not have an oracle-polynomial time algorithm for computing a local optimum. In fact, no polynomial-time algorithm for finding a local optimum is known for any of the above-mentioned problems with neighborhoods of polynomial size,[9] nor for many others. It is instructive to go back to the proof of Lemma 5 and see where it fails when one replaces "augmentation oracle" with "local augmentation oracle" and "optimum" with "local optimum." We will now reproduce an argument from [15], which actually shows that a similar result cannot hold, unless one uses additional information.

[8] Linear combinatorial optimization problems and their neighborhoods for which this is the case belong to the complexity class PLS, defined in [11].

[9] Indeed, these problems are PLS-complete; i.e., the existence of a polynomial-time algorithm that finds local optima for any one of them would imply the existence of such an algorithm for all problems in PLS [11, 32].

More specifically, let A be any algorithm for finding a local optimum, which only uses the following information: the objective function vector c, an initial feasible solution $x^0 \in X$, a local augmentation oracle, and a feasibility oracle. The feasibility oracle accepts as input any point $x \in \{0,1\}^n$, and it answers "yes" or "no," depending on whether x belongs to X or not.

Suppose the coefficients of the objective function vector c are $c_i = 2^{i-1}$, for $i = 1, 2, \ldots, n$, and the initial feasible solution is $x^0 = \mathbb{1}$. Neither the neighborhood nor the set of feasible solutions is fully specified in advance; an adversary will adjust them dynamically, according to the oracle calls made by the algorithm. Let us denote the other potential feasible solutions by $x^1, x^2, \ldots, x^{2^n-1} \in \{0,1\}^n$, with the understanding that $cx^1 > cx^2 > \cdots > cx^{2^n-1}$. The neighborhood of a feasible solution x^i consists of x^j, where j is the smallest index larger than i such that x^j is feasible. If there is no feasible solution x^j with $i < j$, then the neighborhood of x^i is empty.

This neighborhood is exact for the objective function vector c specified above, but it is not necessarily exact for other objective function vectors. The adversary acts as follows. If algorithm A calls the feasibility oracle with some point $x^i \in \{0,1\}^n$ that has not yet been proclaimed feasible, then x^i is declared infeasible. If the algorithm calls the local augmentation oracle with input $x^i \in X$ and $d \in \mathbb{Z}^n$, we distinguish two cases. Let $j > i$ be the smallest index such that x^j has not been labeled infeasible. If such an index exists, then x^j is marked as feasible (but A is not told). If such an index does not exist or if $dx^i \leq dx^j$, then the oracle confirms x^i as locally optimal with respect to d. Otherwise, it returns x^j.

We claim that A needs to touch every single point $x^0, x^1, \ldots, x^{2^n-1}$ before it can announce the correct solution. Indeed, if A never uses the feasibility oracle, then x^{2^n-1} is the only local optimum, and the local augmentation oracle with input x^i and d will either return x^{i+1}, or assert that x^i is a local optimum for d. If A does make use of the feasibility oracle, the local augmentation oracle may return feasible solutions x^j with $j - i > 1$, but only if A previously checked all solutions $x^{i+1}, x^{i+2}, \ldots, x^{j-1}$ for feasibility. Moreover, the unique local optimum may be attained by some x^i with $i < 2^n - 1$, but only if A previously called the feasibility oracle for all points $x^{i+1}, x^{i+2}, \ldots, x^{2^n-1}$. In either case, A requires exponential time.

While the prospects of computing a local optimum in oracle-polynomial time are bleak, it is possible to find a solution that is "nearly" locally optimal in polynomial time [15]:

Theorem 28. *Let $x^0 \in X$, $c \in \mathbb{Z}^n_+$, and $\epsilon > 0$ be given. Assume that the set $X \subseteq \{0,1\}^n$ of feasible solutions and the neighborhood function $N : X \to 2^N$ are specified via a local augmentation oracle. Then, there exists an oracle-polynomial time algorithm that computes a solution $x^\epsilon \in X$ such that*

$$cx^\epsilon \leq (1+\epsilon)cx \text{ for all } x \in N(x^\epsilon). \tag{3}$$

A solution x^ϵ that satisfies (3) is called an ϵ-local optimum.

Proof. The algorithm works in phases. Let x^{k-1} be the current feasible solution at the end of the previous phase. At the beginning of phase k, each objective function coefficient is rounded up to the nearest multiple of q, where $q := (cx^{k-1}\epsilon)/(2n(1+\epsilon))$. Let c^k be the resulting objective function vector. In phase k, we call the local augmentation oracle repeatedly, feeding it with c^k and its own output. The k-th phase ends either when the oracle declares the current feasible solution x locally optimal for c^k, or if the current feasible solution x satisfies $cx \le cx^{k-1}/2$, whichever happens first. In the former case, the algorithm terminates and returns $x^\epsilon := x$; in the latter case, $x^k := x$, and we start phase $k+1$.

Assume that the algorithm stops during phase k. Let x^ϵ be the solution returned by the algorithm. We will show that x^ϵ is an ϵ-local optimum. As $c \le c^k$, we have $cx^\epsilon \le c^k x^\epsilon$. Since x^ϵ is locally optimal with respect to c^k, we get $c^k x^\epsilon \le c^k x$, for all $x \in N(x^\epsilon)$. By the definition of c^k, we have $c^k x = \sum_{i=1}^n \lceil \frac{c_i}{q} \rceil q x_i \le \sum_{i=1}^n q(\frac{c_i}{q}+1)x_i \le cx + nq$. The definition of q and the fact that $cx^\epsilon \ge cx^{k-1}/2$ help to make this chain of inequalities complete, yielding $cx^\epsilon \le cx + \frac{\epsilon}{1+\epsilon} cx^\epsilon$, for all $x \in N(x^\epsilon)$. It follows that x^ϵ is an ϵ-local optimum.

As for the running time, note that the objective function values of any two consecutive solutions in any given phase differ by at least q. Hence, there are at most $O(n/\epsilon)$ calls of the oracle within each phase. The number of phases is itself bounded by $O(\log(cx^0))$. Thus, the algorithm runs in oracle-polynomial time. □

One can actually show that the number of oracle calls made by the algorithm is bounded by $O(\epsilon^{-1} n^2 \log n)$. For details on how to prove this bound as well as other aspects of ϵ-local optima, we refer to [15].

6 General Integer Programming

So far, our discussion has revolved around combinatorial optimization problems that can be naturally formulated as 0/1-integer programs. It is natural to wonder which of the results carry forward to general integer programs, in which variables are still integer, but not necessarily binary. Although this topic has received less attention in the literature, there are a few results worth mentioning.

Throughout this section, we assume that upper bounds $u_i \in \mathbb{Z}_+$ on the variables x_i are explicitly known. Thus, a typical feasible region X is of the form $X \subseteq \{0,1,\ldots,u_1\} \times \{0,1,\ldots,u_2\} \times \cdots \times \{0,1,\ldots,u_n\}$. As before, for a given objective function vector $c \in \mathbb{Z}^n$, let $C := \max\{|c_i| : i = 1,2,\ldots,n\}$; in addition, $U := \max\{u_i : i = 1,2,\ldots,n\} + 1$. We also assume that a feasible point $x^0 \in X$ is given.

"MMA $\longrightarrow$ OPT"

The first result concerns sets of feasible solutions X that are given by a maximum mean augmentation oracle. Let x^* be an optimal solution of $\min\{cx : x \in X\}$. Starting from x^0, we can iteratively call the oracle with input x^i and c. If x^i is not yet optimal, let $x^{i+1} \in X$ be the feasible solution of lower cost returned by the oracle. It is not difficult to show that, in every iteration i, we have $c(x^i - x^{i+1}) \geq c(x^i - x^*)/n$. Hence, we get the following theorem of [24].

Theorem 29. *Let X be given by a maximum mean augmentation oracle. Then, there exists an oracle-polynomial time algorithm that solves the optimization problem for X. The number of calls to the oracle is $O(n \log(nCU))$.*

The authors of [24] consider other maximum ratio oracles as well, including "Wallacher's ratio," which was inspired by barrier functions used in interior-point algorithms for linear programming [30]. This ratio, specified in equation (4) below, is instrumental in proving the next result.

"$\text{AUG}_{\pm}$ $\longrightarrow$ OPT"

Here, the subindex "$\pm$" alludes to a distinction between coordinates that are increased and those that are decreased during augmentation, which is immaterial for 0/1-integer programs. For a vector $z \in \mathbb{Z}^n$, we use z^+ to denote its positive part, and z^- for its negative part. That is, $z_i^+ := \max\{z_i, 0\}$, $z_i^- := \max\{-z_i, 0\}$, for $i = 1, 2, \ldots, n$, and, therefore, $z = z^+ - z^-$. We can now define the following refinement of the augmentation problem, which was introduced in [24].

Directed Augmentation Problem ($\text{AUG}_{\pm}$). Given a feasible solution $x \in X$, and two vectors $c, d \in \mathbb{Z}^n$, find a direction $z \in \mathbb{Z}^n$ such that $x + z \in X$ and $cz^+ + dz^- < 0$, or assert that such a vector z does not exist.

For example, in min-cost flow problems, the directed augmentation problem corresponds to the ordinary augmentation problem for the residual graph.

For any feasible solution $x \in X$ and any coordinate $i \in \{1, 2, \ldots, n\}$, we define $p_i(x) := 1/(u_i - x_i)$ and $n_i(x) := 1/x_i$, with the understanding that $1/0 = \infty$. Binary search can be used to solve the following maximum ratio augmentation problem in oracle-polynomial time, given a directed augmentation oracle.

Maximum Ratio Augmentation Problem (MRA). Given a feasible solution $x \in X$, and a vector $c \in \mathbb{Z}^n$, find a direction $z = z^+ - z^-$ such that $x + z \in X$, $cz < 0$, and z maximizes the ratio

$$\frac{|cz|}{p(x)z^+ + n(x)z^-}, \tag{4}$$

or assert that x is optimal.

For the following result, originally proved in [24], we need the additional technical assumption that X is of the form $X = \{x \in \mathbb{Z}^n : Ax = b, 0 \leq x \leq u\}$, even though neither the matrix A nor the vector b need to be known explicitly.

Theorem 30. *Let X be given by a directed augmentation oracle. Then, there exists an algorithm that solves the optimization problem for X in oracle-polynomial time.*

Proof. We have already sketched the reduction from the maximum ratio augmentation problem to the directed augmentation problem. It remains to show that the optimization problem can be solved in oracle-polynomial time with the help of a maximum ratio augmentation oracle. The algorithm is quite similar to the one used in the proof of Theorem 29. That is, we simply call the maximum ratio oracle repeatedly, feeding it with its own output from the previous iteration. The only difference is that we "stretch" the feasible direction z returned by the oracle, if necessary. Namely, we make sure that $x + z$ is feasible, but $x + 2z$ is not. Note that this implies that there exists a coordinate i such that either $z_i^+ > (u_i - x_i)/2$, or $z_i^- > x_i/2$. Moreover, if z maximizes the ratio (4), then so does any positive multiple of z.

To analyze the running time of this algorithm, let x be the feasible point at the start of the current iteration, and let x^* be an optimal solution. Moreover, let $z^* := x^* - x$. Using the previous observation and $p(x)(z^*)^+ + n(x)(z^*)^- \leq n$, we obtain that $|cz| \geq |cz^*|/(2n)$. Since the difference in objective function values between the initial feasible solution x^0 and the optimal solution x^* is $O(nCU)$, it follows that the algorithm terminates with an optimal solution after $O(n \log(nCU))$ calls of the maximum ratio oracle. □

In total, one needs $O(n^2 \log^2(nCU))$ calls of the directed augmentation oracle to solve the optimization problem. A more efficient algorithm, which gets by with $O(n \log(nCU))$ oracle calls, is described in [24].

INC ⟶ OPT and EVA ⟶ OPT

Interestingly, Lemma 13 and its proof remain valid for general integer programs.[10] Consequently, given a unit increment oracle, one can solve the optimization problem in oracle polynomial time. In fact, one can show something stronger. To facilitate the discussion, we restrict ourselves to nonnegative objective function vectors and introduce the following evaluation version of the unit increment problem.

[10] The necessary modification of the switch operation for coordinates with negative objective function coefficients is straightforward: $x_i \mapsto u_i - x_i$.

Unit Increment Evaluation Problem (INC-EVA). Given a vector $c \in \mathbb{Z}_+^n$, and an index $i \in \{1, 2, \ldots, n\}$, compute the difference of the optimal objective function values for $\min\{(c + e_i)x : x \in X\}$ and $\min\{cx : x \in X\}$.

Accordingly, the output is a single number. Obviously, given an oracle for the evaluation problem, one can solve the unit increment evaluation problem in oracle-polynomial time. In fact, the reverse is also true, as can easily be derived from the proof of Lemma 13. The following theorem, taken from [16], shows that either oracle suffices to solve the optimization version of general integer programming problems.

Theorem 31. *Assume that X is given by a (unit increment) evaluation oracle. Then one can solve the optimization problem for X in oracle-polynomial time.*

Proof. Here is an outline of the proof. Given $c \in \mathbb{Z}_+^n$, let y be the lexicographically smallest optimal solution of $\min\{cx : x \in X\}$. We define a new objective function vector d that has the following properties:

(a) The solution y is optimal with respect to d.
(b) The solution y is optimal with respect to $d + e_i$, for all $i = 1, 2, \ldots, n$.

Obviously, if this is the case, one can use the oracle to compute $(d+e_i)y - dy$, and recover y via $y_i = (d + e_i)y - dy$, for $i = 1, 2, \ldots, n$.

It remains to define the objective function vector d:

$$d_i := nU^{2n}c_i + (n - i + 1)U^{2(n-i)} \text{ for } i = 1, 2, \ldots, n.$$

The verification of (a) and (b) is left to the reader. □

Acknowledgments

The author is grateful to Juliane Dunkel, Berit Johannes, and an anonymous referee for several comments on earlier versions of this chapter, which helped to improve the presentation. He also likes to thank Jens Vygen for all is help and support.

References

1. Ravindra K. Ahuja and James B. Orlin. Inverse optimization. *Operations Research*, 49:771–783, 2001.
2. Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation.* Springer, 1999.
3. Nilotpal Chakravarti and Albert P. M. Wagelmans. Calculation of stability radii for combinatorial optimization problems. *Operations Research Letters*, 23:1–7, 1998.

4. Pierluigi Crescenzi and Riccardo Silvestri. Relative complexity of evaluating the optimum cost and constructing the optimum for maximization problems. *Information Processing Letters*, 33:221–226, 1990.
5. Friedrich Eisenbrand, Giovanni Rinaldi, and Paolo Ventura. Primal separation for 0/1 polytopes. *Mathematical Programming*, 95:475–491, 2003.
6. Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.
7. András Frank and Éva Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.
8. Martin Grötschel and László Lovász. Combinatorial optimization. In Ronald L. Graham, Martin Grötschel, and László Lovász, editors, *Handbook of Combinatorics*, volume 2, chapter 28, pages 1541–1597. Elsevier, 1995.
9. Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.
10. Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.
11. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.
12. Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 4th edition, 2008.
13. Adam N. Letchford and Andrea Lodi. Primal cutting plane algorithms revisited. *Mathematical Methods of Operations Research*, 56:67–81, 2002.
14. Denis Naddef. The Hirsch conjecture is true for $(0, 1)$-polytopes. *Mathematical Programming*, 45:109–110, 1989.
15. James B. Orlin, Abraham P. Punnen, and Andreas S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33:1201–1214, 2004.
16. James B. Orlin, Abraham P. Punnen, and Andreas S. Schulz. In preparation, 2008.
17. Manfred W. Padberg and Martin Grötschel. Polyhedral computations. In Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, chapter 9, pages 307–360. Wiley, 1985.
18. Manfred W. Padberg and Saman Hong. On the symmetric travelling salesman problem: A computational study. *Mathematical Programming Study*, 12:78–107, 1980.
19. Christos H. Papadimitriou and Kenneth Steiglitz. On the complexity of local search for the traveling salesman problem. *SIAM Journal on Computing*, 6:76–83, 1977.
20. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
21. Ramkumar Ramaswamy and Nilotpal Chakravarti. Complexity of determining exact tolerances for min-sum and min-max combinatorial optimization problems. Working Paper WPS-247/95, Indian Institute of Management, Calcutta, India, 1995.
22. Sam Savage, Peter Weiner, and Amitava Bagchi. Neighborhood search algorithms for guaranteeing optimal traveling salesman tours must be inefficient. *Journal of Computer and System Sciences*, 12:25–35, 1976.

23. Sam L. Savage. *The Solution of Discrete Linear Optimization Problems by Neighborhood Search Techniques*. Doctoral dissertation, Yale University, New Haven, CT, 1973.
24. Andreas S. Schulz and Robert Weismantel. The complexity of generic primal algorithms for solving general integer programs. *Mathematics of Operations Research*, 27:681–692, 2002.
25. Andreas S. Schulz, Robert Weismantel, and Günter M. Ziegler. 0/1-integer programming: Optimization and augmentation are equivalent. *Lecture Notes in Computer Science*, 979:473–483, 1995.
26. Andreas S. Schulz, Robert Weismantel, and Günter M. Ziegler. An optimization problem is nine problems. Talk presented by Andreas S. Schulz at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, 1997.
27. Craig A. Tovey. Hill climbing with multiple local optima. *SIAM Journal on Algebraic and Discrete Methods*, 6:384–393, 1985.
28. Stan van Hoesel and Albert Wagelmans. On the complexity of postoptimality analysis of 0/1 programs. *Discrete Applied Mathematics*, 91:251–263, 1999.
29. Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.
30. Claus Wallacher. *Kombinatorische Algorithmen für Flußprobleme und submodulare Flußprobleme*. Doctoral dissertation, Technische Universität Carolo-Wilhelmina zu Braunschweig, Germany, 1992.
31. Robert Weismantel. Test sets of integer programs. *Mathematical Methods of Operations Research*, 47:1–37, 1998.
32. Mihalis Yannakakis. Computational complexity. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 2, pages 19–55. Wiley, 1997.
33. R. D. Young. A simplified primal (all-integer) integer programming algorithm. *Operations Research*, 16:750–782, 1968.
34. Günter M. Ziegler. *Lectures on Polytopes*. Springer, 1995.