0/1–Integer Programming: Optimization and Augmentation are Equivalent

Andreas S. Schulz* Robert Weismantel† Günter M. Ziegler‡

March 1995

Abstract

For every fixed set $\mathcal{F} \subseteq \{0,1\}^n$ the following problems are strongly polynomial time equivalent: given a feasible point $x \in \mathcal{F}$ and a linear objective function $c \in \mathbb{Z}^n$,

- find a feasible point $x^* \in \mathcal{F}$ that maximizes cx (Optimization),
- find a feasible point $x^{\text{new}} \in \mathcal{F}$ with $cx^{\text{new}} > cx$ (Augmentation), and
- find a feasible point $x^{\text{new}} \in \mathcal{F}$ with $cx^{\text{new}} > cx$ such that $x^{\text{new}} - x$ is "irreducible" (Irreducible Augmentation).

This generalizes results and techniques that are well known for 0/1–integer programming problems that arise from various classes of combinatorial optimization problems.

1 Introduction

For any fixed set $\mathcal{F} \subseteq \{0,1\}^n$ of *feasible* 0/1–points, we show that optimization and (irreducible) augmentation with respect to linear objective functions are strongly polynomial time equivalent. For example, $\mathcal{F}$ can be given as $\mathcal{F} = \{x \in \{0,1\}^n : Ax \leq b\}$ for a fixed matrix $A \in \mathbb{Z}^{m \times n}$ and a fixed vector $b \in \mathbb{Z}^m$, but such an explicit representation need not be given for the following. However, we assume throughout that some *feasible* solution $x \in \mathcal{F}$ is known in advance. The optimization problem for $\mathcal{F}$ is the following task.

The Optimization Problem (OPT)
Given a vector $c \in \mathbb{Z}^n$, find a vector $x^* \in \mathcal{F}$ that maximizes cx on $\mathcal{F}$.

*Technische Universität Berlin, Fachbereich Mathematik (MA 6–1), Straße des 17. Juni 136, D–10623 Berlin, Germany, schulz@math.tu–berlin.de

†Konrad–Zuse–Zentrum für Informationstechnik Berlin, Heilbronner Straße 10, D–10711 Berlin, Germany, weismantel@zib–berlin.de

‡Technische Universität Berlin, Fachbereich Mathematik (MA 6–1), Straße des 17. Juni 136, D–10623 Berlin, Germany, ziegler@math.tu–berlin.de

Many combinatorial optimization problems, such as unit capacity network flow and matching problems, can be described in this form. One can also model linear optimization problems for integral points $x \in \mathbb{Z}^n$ in this way, as long as there are bounds $0 \leq x_j \leq u_j$ for the variables $x \in \mathbb{Z}^n$ that are polynomially bounded in the input size. For this, just replace x_j by $x_{j1} + x_{j2} + \cdots + x_{ju_j}$, where the x_{j_i} are 0/1–variables.

Several known polynomial time algorithms for solving 0/1–optimization problems are of primal nature. That is, given a feasible solution, i. e., a point $x^0 \in \mathcal{F}$, one successively produces new feasible solutions $x^1, x^2, \ldots$ with $cx^0 < cx^1 < cx^2 < \cdots$ until an optimal solution is reached. From an abstract point of view an *augmentation problem* is solved in each iteration: for given x^k find an *augmentation vector* z such that $cz > 0$, and $x^k + z$ is feasible.

The Augmentation Problem (AUG)
Given a vector $c \in \mathbb{Z}^n$ and a point $x^{\text{old}} \in \mathcal{F}$, find a point $x^{\text{new}} \in \mathcal{F}$ such that $cx^{\text{new}} > cx^{\text{old}}$, or assert that no such x^{new} exists.

Most primal algorithms, however, need to make a special choice of the augmentation vector $x^{\text{new}} - x^{\text{old}}$ in each iteration, in order to come up with a polynomial number of overall iterations. In cycle–canceling algorithms for the min cost flow problem, for instance (see, e.g., [AMO93]), in each iteration flow has to be augmented along a negative cycle with minimum (mean) cost.

Somehow surprisingly, we show that (OPT) can be solved by calling a strongly polynomial number of times an oracle that solves (AUG). The surprise is certainly not only due to the fact that this holds for *arbitrary* 0/1–programs but also that it is sufficient to be able to find an arbitrary augmentation vector, for any integral objective function. We do not need the best one with respect to a certain measure.

Observe that there is another difference to cycle–canceling algorithms. In these algorithms we restrict to simple cycles (irreducible augmentation vectors), while in (AUG) arbitrary augmentation vectors are allowed for. We call an augmentation vector z *reducible* if there exist two vectors $v, w \neq 0$ such that

- $v + w = z$,
- $v^+, w^+ \leq z^+$,
- $v^-, w^- \leq z^-$, and
- $x^{\text{old}} + v, x^{\text{old}} + w \in \mathcal{F}$.

In this situation v or w is a "smaller" augmentation vector that can be applied instead of z at the point x^{old}. Here, for a vector y we denote by y^+ and y^- its positive and negative parts, respectively, i. e., $y_j^+ := \max\{0, y_j\}$, and $y_j^- := -\min\{0, y_j\}$. Thus, $y = y^+ - y^-$. In case z is not reducible, it is called *irreducible*. Notice that v, w, and z are augmentation vectors with respect to the same point x^{old}.

The Irreducible Augmentation Problem (IRR–AUG)
Given a vector $c \in \mathbb{Z}^n$ and a point $x^{\text{old}} \in \mathcal{F}$, find a point $x^{\text{new}} \in \mathcal{F}$ such that $x^{\text{new}} - x^{\text{old}}$ is irreducible and $cx^{\text{new}} > cx^{\text{old}}$, or assert that no such x^{new} exists.

The restriction to irreducible augmentation vectors does not affect the existence of a strongly oracle–polynomial time algorithm solving (OPT) when $\mathcal{F}$ is given by an oracle for (IRR–AUG). In fact, whereas (AUG) and (IRR–AUG) have the same input, each solution of (IRR–AUG) is a solution of (AUG), but not vice versa. It is by no means trivial to solve the irreducible augmentation problem given an oracle for solving (OPT), or, equivalently, (AUG).

Theorem 1.1
Any one of the following three problems:

- *optimization* (OPT),
- *augmentation* (AUG),
- *irreducible augmentation* (IRR–AUG),

can be solved in strongly oracle–polynomial time for any set $\mathcal{F} \subseteq \{0,1\}^n$ given by an oracle for any of the other two problems.

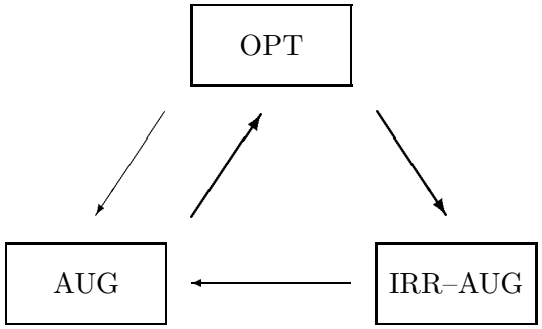

Figure 1

Figure 1 indicates the trivial relations between the three problems by thin arrows. Here, an arrow means that the problem at the head of the arrow can be solved in strongly oracle–polynomial time given an oracle for the problem at the tail of the arrow. The two thick arrows represent our main results that are presented in Sections 2 and 3, respectively. In both cases we first derive a polynomial time algorithm, and then use the "preprocessing algorithm" of Frank and Tardos [FT87] to turn it into a strongly polynomial procedure.

For a thorough introduction to oracles and oracle–polynomial time algorithms as well as strongly polynomial time algorithms we refer to Grötschel, Lovász, and Schrijver [GLS88], see also Lovász [Lov86].

2 An Oracle–Polynomial Time Algorithm for Optimization

In this section we state and analyze an oracle–polynomial time algorithm that solves (OPT), assuming an oracle for (AUG) is given. We first concentrate on nonnegative objective function vectors but shall conclude with a discussion of the transformation needed to allow arbitrary objectives. Finally, we point out how to turn this into a strongly polynomial time algorithm.

The essential idea underlying our algorithm is *bit–scaling* (see [EK72]). Thus we have to represent data by binary numbers. For $\alpha \in \mathbb{N}$ and a given number K that is at least as big as the number of bits needed to encode α, i. e., $K \geq \lceil \log(\alpha + 1) \rceil$, we represent α as a K–bit binary number, adding leading zeros if necessary. We denote by $\alpha(k)$ the number obtained by considering the k leading bits only. With ${}^k\alpha$ we refer to the k–th bit of α. Thus, $\alpha(k) = ({}^1\alpha, {}^2\alpha, \ldots, {}^k\alpha) = \sum_{i=1}^{k} {}^i\alpha 2^{k-i}$, and $\alpha(K) = \alpha$.

Scaling methods have extensively been used to derive polynomial time algorithms for a wide variety of network and combinatorial optimization problems (see, e.g., [AMO93]). In this section we use bit–scaling of costs to derive an oracle–polynomial time algorithm for optimizing 0/1–integer programs. This technique has been used earlier by Röck [Rö80] and Gabow [Gab85] for solving minimum cost flow and shortest path problems, respectively.

As already mentioned we assume first that all coefficients of the objective function vector are nonnegative.

> **The Optimization Problem (OPT)$^{\geq 0}$**
> Given a vector $c \in \mathbb{N}^n$, find a vector $x^* \in \mathcal{F}$ that maximizes cx on $\mathcal{F}$.

Restricting the optimization problem in this way, it seems to be reasonable to do the same with the augmentation problem. That is, we restrict it to nonnegative input vectors c, too.

> **The Augmentation Problem (AUG)$^{\geq 0}$**
> Given a vector $c \in \mathbb{N}^n$ and a point $x^{\text{old}} \in \mathcal{F}$, find a point $x^{\text{new}} \in \mathcal{F}$ such that $cx^{\text{new}} > cx^{\text{old}}$, or assert that no such x^{new} exists.

Theorem 2.1 *There exists an algorithm that solves* $(\text{OPT})^{\geq 0}$ *by* $\mathcal{O}(n \log C)$ *calls of an oracle that solves* $(\text{AUG})^{\geq 0}$*, for* $C = \max\{c_j : j = 1, \ldots, n\} + 1$.

Proof. Let $K = \lceil \log C \rceil$. We present a bit–scaling algorithm solving a sequence of problems $(\text{P}_1), (\text{P}_2), \ldots, (\text{P}_K)$. The objective of (P_1) consists of the most significant bit only, the one of (P_2) of the first two most significant bits, and so on. Problem (P_K) will be the original problem to be solved.

For $k = 1, \ldots, K$ we define (P_k) as

$$\begin{array}{ll} \max & c(k)x \\ \text{s. t.} & x \in \mathcal{F} \end{array}$$

Here, $c(k) \in \mathbb{N}^n$ denotes the vector obtained from c by restricting each component to the k leading bits, $c(k) = (c_1(k), \ldots, c_n(k))$.

Algorithm 2.2 (Bit–scaling algorithm for solving $(\mathrm{OPT})^{\geq 0}$)

1. let x^0 be a feasible solution;
2. $k := 1$;
3. while $k \leq K$ do
4. solve (P_k) by iterated use of $(\mathrm{AUG})^{\geq 0}$, with initial solution x^{k-1};
5. let x^k be the optimal solution of (P_k);
6. $k := k + 1$;
7. end.

Step 4 of Algorithm 2.2 needs further explanation. The general idea is to start with a feasible solution y^0, say, to call the augmentation oracle with y^0, and to obtain a better solution y^1 that serves as the new input, and so forth. Since (P_k) is bounded, the procedure is finite. To keep the number of these inner iterations small it is important to use x^{k-1} as the starting solution.

Since (P_K) coincides with the original optimization problem Algorithm 2.2 is correct. It remains to be shown that the number of calls of $(\mathrm{AUG})^{\geq 0}$ in Step 4 to determine x^k starting from x^{k-1} is polynomially bounded in the input size. The following calculation shows that in Step 4 for each problem (P_k) the oracle for $(\mathrm{AUG})^{\geq 0}$ is called at most n times:

$$c(k)(x^k - x^{k-1}) = 2c(k-1)(x^k - x^{k-1}) + {}^k c\,(x^k - x^{k-1}) \leq 0 + n.$$

The inequality follows from the optimality of x^{k-1} for (P_{k-1}), from ${}^k c \in \{0,1\}^n$, and from $x^k, x^{k-1} \in \{0,1\}^n$. (We define $c(0)$ to be zero.) □

The last obstacle on our way to an oracle–polynomial time algorithm for 0/1–programming is to get rid of the assumption on the objective function vectors. We made this nonnegativity assumption in order to simplify the bit–scaling. We shall present an easy transformation from (OPT) to $(\mathrm{OPT})^{\geq 0}$ as well as from (AUG) to $(\mathrm{AUG})^{\geq 0}$. Given an instance of (OPT),

$$\begin{array}{ll} \max & cx \\ \text{s. t.} & x \in \mathcal{F} \end{array}$$

with $c \in \mathbb{Z}^n$, we define an instance of $(\mathrm{OPT})^{\geq 0}$ as follows. Let $\tilde{c} \in \mathbb{N}^n$ be the vector with coefficients

$$\tilde{c}_j = \begin{cases} c_j, & \text{if } c_j \geq 0, \\ -c_j, & \text{otherwise,} \end{cases}$$

and let for $x \in \mathcal{F}$

$$\tilde{x}_j = \begin{cases} x_j, & \text{if } c_j \geq 0, \\ 1 - x_j, & \text{otherwise.} \end{cases}$$

With $\widetilde{\mathcal{F}} := \{\tilde{x} : x \in \mathcal{F}\}$ the following defines an instance of $(\text{OPT})^{\geq 0}$:

$$\begin{array}{ll} \max & \tilde{c}\,\tilde{x} \\ \text{s. t.} & \tilde{x} \in \widetilde{\mathcal{F}} \end{array}$$

Then $x \in \mathcal{F}$ is optimal with respect to c if and only if $\tilde{x} \in \widetilde{\mathcal{F}}$ is optimal with respect to $\tilde{c}$.

From the discussion above, we know that $(\text{OPT})^{\geq 0}$ can be solved in polynomial time assuming an oracle for solving $(\text{AUG})^{\geq 0}$ is given. Since $(\text{AUG})^{\geq 0}$ for $\widetilde{\mathcal{F}}$ can be solved by calling the (AUG) oracle for $\mathcal{F}$, we obtain a polynomial time algorithm for optimization in terms of an augmentation oracle, as follows.

Corollary 2.3 *There exists an algorithm that solves* (OPT) *by* $\mathcal{O}(n \log C)$ *calls of an oracle that solves* (AUG)*, for* $C = \max\{|c_j| : j = 1, \ldots, n\} + 1$.

Using the "preprocessing algorithm" of Frank and Tardos [FT87], we turn this into a strongly polynomial algorithm. Namely, in time polynomial in n we replace the original objective function c by a new integral objective function $\overline{c}$ whose size is polynomially bounded in n, such that $\text{sign}(cv) = \text{sign}(\overline{c}v)$ holds for all vectors $v \in \{+1, 0, -1\}^n$, and thus for all possible augmentation vectors (differences of feasible points). Then the algorithm just described is run for the new objective function $\overline{c}$.

This completes the proof of the implication (AUG)$\longrightarrow$(OPT) of Theorem 1.1.

3 An Oracle–Polynomial Time Algorithm for the Irreducible Augmentation Problem

This section is devoted to showing that (IRR–AUG) can be solved in (strongly) polynomial time, assuming an oracle that solves (OPT) is given. We divide this problem into two parts. First, we show how to determine a maximum mean augmentation vector, i. e., an augmentation vector with the maximum ratio of improvement to cardinality of support. Since there exists an irreducible augmentation vector attaining this optimum value, we shall then provide an algorithm to determine such an augmentation vector. Finally, we use again the preprocessing algorithm of Frank and Tardos to turn this into a strongly polynomial time algorithm.

One remark shall be given in advance. In the formulation of (OPT) we assumed the objective function vector c to be integer valued since (starting from rational numbers) this can always be achieved by appropriate scaling. In the following we will construct objective functions that are not integer valued. This is only done for simplifying the presentation. In these cases we always assume the scaling to be performed implicitly. One should observe, however, that this scaling can indeed be done without affecting the magnitude of the size of the objective function.

3.1 The Maximum Mean Augmentation Problem

One step towards solving the irreducible augmentation problem via a polynomial number of calls of the optimization oracle is to find the maximum mean augmentation vector. This

question is addressed in this section. The technique that we use is quite similar to the one used for the *minimum cost–to–time ratio cycle problem* (see [AMO93, pp. 150–152]).

The Maximum Mean Augmentation Problem (MMA)
Given a vector $c \in \mathbb{Z}^n$ and a point $x^{\text{old}} \in \mathcal{F}$, find a point $x^{\text{new}} \in \mathcal{F}$ such that $cx^{\text{new}} > cx^{\text{old}}$ and x^{new} maximizes $\frac{cx - cx^{\text{old}}}{|x - x^{\text{old}}|_1}$ over $\mathcal{F} \setminus \{x^{\text{old}}\}$, or assert that no such x^{new} exists.

Here, $|z|_1$ denotes the L_1–norm of z, which coincides with the cardinality of the support of an augmentation vector. Throughout this section we assume that $c \in \mathbb{Z}^n$ and $x^{\text{old}} \in \mathcal{F}$ are given. By S^{old} we denote the support of the vector x^{old}, i. e., $S^{\text{old}} := \{j \in N : \ x_j^{\text{old}} = 1\}$.

In order to find a maximum mean augmentation vector we proceed as follows. We first call (OPT) with the linear functional c. In case that x^{old} is optimal, there does not exist an augmentation vector. Thus, in the following we assume that x^{old} is not optimal with respect to c. Let μ^* denote the optimal objective function value of (MMA). For any arbitrary value $0 < \mu \leq 2\sum_{j \in N} |c_j|$ we define an objective function c^μ as follows:

$$c_j^\mu := \begin{cases} c_j + \mu, & \text{if } j \in S^{\text{old}}, \\ c_j - \mu, & \text{otherwise.} \end{cases}$$

Calling (OPT) with c^μ as input, we distinguish three different situations. Let x^μ denote the output of (OPT).

Case 1. $x^\mu = x^{\text{old}}$.

In this case, $c^\mu x \leq c^\mu x^{\text{old}}$ for every $x \in \mathcal{F} \setminus \{x^{\text{old}}\}$. Alternatively,

$$\frac{c(x - x^{\text{old}})}{|x - x^{\text{old}}|_1} \leq \mu.$$

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

Case 2. $x^\mu \neq x^{\text{old}}$ and $c^\mu x^\mu = c^\mu x^{\text{old}}$.

As in the previous case we obtain

$$\frac{c(x - x^{\text{old}})}{|x - x^{\text{old}}|_1} \leq \mu, \quad \text{for all } x \in \mathcal{F} \setminus \{x^{\text{old}}\}.$$

Since x^μ satisfies this inequality with equality, $\mu = \mu^*$.

Case 3. $x^\mu \neq x^{\text{old}}$ and $c^\mu x^\mu > c^\mu x^{\text{old}}$.

In this case,

$$\mu^* \geq \frac{c(x^\mu - x^{\text{old}})}{|x^\mu - x^{\text{old}}|_1} > \mu.$$

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

Based on the preceding case analysis we can determine μ^* by binary search. Observe that μ^* lies in the interval $(0, 2\sum_{j\in N}|c_j|]$. Thus, we start with the lower bound $\underline{\mu} = 0$ and the upper bound $\overline{\mu} = 2\sum_{j\in N}|c_j|$. At every iteration we consider $\mu = (\underline{\mu} + \overline{\mu})/2$ and call (OPT) with input c^μ. If we encounter Case 1 or 3 we reset $\overline{\mu} = \mu$ and $\underline{\mu} = \mu$, respectively, whereas in Case 2 we are done. At every iteration, we halve the length of the search interval. Given any two augmentation vectors with distinct ratios the absolute difference between their ratios is at least $1/n^2$. Hence, if $\overline{\mu} - \underline{\mu} < 1/n^2$, the current interval $[\underline{\mu}, \overline{\mu}]$ contains at most one ratio of the form $\frac{cz}{|z|_1}$ where z is an augmentation vector. Consequently, calling (OPT) again with $\underline{\mu}$ we either obtain $\mu^* = \underline{\mu}$ but $x^{\mu^*} = x^{\text{old}}$ (Case 1), or we obtain $\mu^* = \underline{\mu}$ and x^{μ^*} immediately (Case 2), or, in Case 3, we obtain x^{μ^*} from which we can also compute μ^*. In the first case, we still have to determine $x^{\text{new}} \neq x^{\text{old}}$ such that

$$\frac{c(x^{\text{new}} - x^{\text{old}})}{|x^{\text{new}} - x^{\text{old}}|_1} = \mu^*.$$

This can be done as follows. Set $M := 2(2n+1)\sum_{j\in N}|c_j| + 1$. For every $i \in N$, we define an objective function d^i by setting

$$d^i_j := \begin{cases} c^{\mu^*}_j + (-1)^k M, & \text{if } i = j, \\ c^{\mu^*}_j, & \text{otherwise,} \end{cases}$$

with $k = 1$ if $i \in S^{\text{old}}$, and $k = 2$, otherwise. For every $i \in N$, we call (OPT) with objective function d^i, and denote by y^i the point that is returned by (OPT). Among all these points y^i, $i \in N$, there is one y^*, say, such that $y^* \neq x^{\text{old}}$ and $\frac{cy^* - cx^{\text{old}}}{|y^* - x^{\text{old}}|_1} = \mu^*$, because there exists $y \neq x^{old}$ such that $\frac{cy - cx^{\text{old}}}{|y - x^{old}|_1} = \mu^*$, and $y \neq x^{\text{old}}$ implies that there exists an index $i^* \in N$ with $y_{i^*} \neq x^{\text{old}}_{i^*}$.

Corollary 3.1 *There exists an algorithm that solves* (MMA) *by* $\mathcal{O}(n + \log(n\,C))$ *calls of an oracle that solves* (OPT).

3.2 Determining an Irreducible Augmentation Vector

In the preceding section we showed how to determine a maximum mean augmentation vector, assuming an optimization oracle is given. We shall now show that there exists an irreducible augmentation vector sharing this property. This will enable us to determine such a vector. This completes the proof that (IRR–AUG) can be solved by calling an oracle for (OPT) a polynomial number of times. We continue using the notation introduced in the previous section.

Proposition 3.2 *Assume that* $x \in \mathcal{F}\backslash\{x^{\text{old}}\}$ *is a point such that* $\frac{cx - cx^{\text{old}}}{|x - x^{\text{old}}|_1} = \mu^*$. *If* $z := x - x^{\text{old}}$ *is reducible,* $z = v + w$, *say, then* $\frac{cv}{|v|_1} = \frac{cw}{|w|_1} = \mu^*$.

Proof. Since z is reducible by v and w, we obtain $|v + w|_1 = |v|_1 + |w|_1$. Together with $\frac{c(v+w)}{|v+w|_1} \geq \frac{cv}{|v|_1}$ and $\frac{c(v+w)}{|v+w|_1} \geq \frac{cw}{|w|_1}$, this implies $\frac{cv}{|v|_1} = \frac{cw}{|w|_1}$. We conclude that $\frac{c(v+w)}{|v+w|_1} = \frac{cv}{|v|_1} = \frac{cw}{|w|_1}$. □

We use Proposition 3.2 to compute an irreducible maximum mean augmentation vector: if z is reducible by v and w, the support of both v and w is properly contained in the support of z. We exploit this by appropriate perturbation of the objective function.

Let x^{μ^*} still denote the output of (MMA), and let $z := x^{\mu^*} - x^{\text{old}}$ be the associated augmentation vector. Observe that there always exists an index $i^* \in N$ such that $z_{i^*} = (-1)^k$ for some $k \in \{1, 2\}$, and x^{μ^*} is optimal with respect to the objective function d^{i^*} whereas x^{old} is not. If z is reducible, $z = v + w$, we may assume that $v_{i^*} = (-1)^k$. Therefore, Proposition 3.2 implies that $x^{\text{old}} + v$ is also optimal with respect to d^{i^*}. We now describe the appropriate perturbation of d^{i^*} in order to end up (by calling (OPT) with the perturbed function) with an irreducible augmentation vector. For two points $y, y' \in \mathcal{F}$ that have two different objective function values the difference $|d^{i^*}y - d^{i^*}y'|$ is at least $\frac{1}{n}$ because μ^* has denominator at most n. Setting $\epsilon := \frac{1}{1+2n^2}$, we define the perturbed objective function $\tilde{d}^{i^*}$ as follows:

$$\tilde{d}^{i^*}_j := \begin{cases} d^{i^*}_j - \epsilon & \text{if } j \in N \setminus S^{\text{old}}, \\ d^{i^*}_j + \epsilon & \text{if } j \in S^{\text{old}}. \end{cases}$$

For every point $y \in \mathcal{F}$ we have

$$\begin{aligned} \tilde{d}^{i^*}y &= d^{i^*}y + \epsilon(|S^{\text{old}}| - |\{j \in S^{\text{old}} : \ y_j = 0\}| - |\{j \in N \setminus S^{\text{old}} : \ y_j = 1\}|) \\ &= d^{i^*}y + \epsilon|S^{\text{old}}| - \epsilon|y - x^{\text{old}}|_1 \\ &= c^{\mu^*}y + (-1)^k M y_{i^*} + \epsilon|S^{\text{old}}| - \epsilon|y - x^{\text{old}}|_1. \end{aligned}$$

Now, observe that by the choice of ϵ every point $y \in \mathcal{F}$ that is optimal with respect to $\tilde{d}^{i^*}$ is also optimal with respect to d^{i^*}. Therefore, (OPT) will return a vector $x^{\text{old}} + v$ with $\frac{cv}{|v|_1} = \mu^*$ and $v_{i^*} = (-1)^k$. The vector v is irreducible, for if not, then there would exist a vector w such that $x^{\text{old}} + w \in \mathcal{F}$, $\frac{cw}{|w|_1} = \mu^*$, $w_{i^*} = (-1)^k$, and $|w|_1 < |v|_1$. The latter fact would imply that $\tilde{d}^{i^*}w > \tilde{d}^{i^*}v$, a contradiction.

The following theorem summarizes what we have obtained so far.

Theorem 3.3 *There exists an algorithm that solves* (IRR–AUG) *by* $\mathcal{O}(n + \log(n\,C))$ *calls of an oracle that solves* (OPT).

Again, using the "preprocessing algorithm" of Frank and Tardos [FT87], we turn this into a strongly polynomial time algorithm.

This time, the "quality" of the preprocessing is chosen to be good enough to guarantee that $\text{sign}(cv) = \text{sign}(\overline{c}v)$ holds for all vectors $v \in \mathbb{Z}^n$ with $|v_j| \leq n$. This implies that not only a vector $v \in \{+1, 0, -1\}^n$ is an augmentation vector with respect to the original objective function c if and only if it is an augmentation vector with respect to $\overline{c}$, but also that the maximum mean augmentation vectors with respect to c coincide with those for $\overline{c}$. Namely, x provides a better mean augmentation than x' if and only if

$$c\Big((x - x^{\text{old}})\,|x' - x^{\text{old}}|_1\Big) \ > \ c\Big((x' - x^{\text{old}})\,|x - x^{\text{old}}|_1\Big),$$

and by Frank and Tardos this is equivalent to

$$\overline{c}\Big((x - x^{\text{old}})\,|x' - x^{\text{old}}|_1\Big) \ > \ \overline{c}\Big((x' - x^{\text{old}})\,|x - x^{\text{old}}|_1\Big).$$

Now we can run the polynomial time algorithm described above on the objective function $\overline{c}$, and obtain a strongly polynomial time procedure for the implication (OPT)$\longrightarrow$(IRR–AUG) of Theorem 1.1. The output is, in particular, a maximum mean augmentation vector with respect to the original objective function c.

4 Concluding Remarks

We have established the equivalence between the optimization and the augmentation problem for $\mathcal{F}$ with respect to strongly polynomial time solvability. The algorithm presented for solving (OPT) given an oracle for (AUG) does not only generalize some known algorithms for special combinatorial problems, but is also a new one for treating some of them. Its essence is that a problem is polynomially tractable if and only if for any objective function we are efficiently able to find always a new feasible solution that improves upon the current one.

On the other hand, we cannot expect to solve (AUG) in polynomial time if the corresponding optimization problem is $\mathcal{NP}$–hard, unless $\mathcal{P}$=$\mathcal{NP}$. We may hope however, to be able to solve (AUG) efficiently if we restrict ourselves to special augmentation vectors. Then, of course, by the algorithm described above we will not necessarily obtain an optimal solution, but maybe a good one. The equivalence of (AUG) and (IRR–AUG) suggests to search for irreducible augmentation vectors.

Finally, let us point out the relation to test sets in integer programming (see, e.g., [Sch86]). Given an oracle that solves (AUG) we have access to a test set for the given integer program. Similarly, given an oracle for (IRR–AUG) we are implicitly given a *minimal* test set. From this point of view, the results presented in this paper imply in particular that for 0/1–programming problems optimization and augmentation by use of (minimal) test sets are equivalent in terms of computational complexity.

Acknowledgements

Thanks to Lex Schrijver for an important augmentation vector for this paper.

References

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin: *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs NJ, 1993.

[EK72] Jack Edmonds and Richard M. Karp: Theoretical improvements in algorithmic efficiency for network flow problems, *Journal of the Association for Computing Machinery* 19 (1972), 248–264.

[FT87] András Frank and Éva Tardos: An application of simultaneous Diophantine approximation in combinatorial optimization, *Combinatorica* 7 (1987), 49–65.

[Gab85] Harold N. Gabow: Scaling algorithms for network problems, *Journal of Computer and System Sciences* 31 (1985), 148–168.

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver: *Geometric Algorithms and Combinatorial Optimization*, Algorithms and Combinatorics 2, Springer, Berlin, 1988; Second edition 1993.

[Lov86] László Lovász: *An Algorithmic Theory of Numbers, Graphs and Convexity*, CBMS-NSF Regional Conference Series in Applied Mathematics 50, Society for Industrial and Applied Mathematics (SIAM), Philadelphia 1986.

[Rö80] Hans Röck: Scaling techniques for minimal cost network flows, in: V. Page (ed.), *Discrete Structures and Algorithms*, Carl Hanser, Munich, 1980, pp. 181–191.

[Sch86] Alexander Schrijver: *Theory of Linear and Integer Programming*, John Wiley & Sons, Chichester, 1986.