A Variant of the Buchberger Algorithm for Integer Programming

Regina Urbaniak Robert Weismantel Günter M. Ziegler*
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
Heilbronner Str. 10, 10711 Berlin, Germany
[last_name]@zib-berlin.de

Abstract

In this paper we modify Buchberger's $\mathcal{S}$-pair reduction algorithm for computing a Gröbner basis of a toric ideal so as to apply to an integer program in inequality form with fixed right hand sides and fixed upper bounds on the variables. We formulate the algorithm in the original space and interpret the reduction steps geometrically. In fact, three variants of this algorithm are presented and we give elementary proofs for their correctness. A relationship between these (exact) algorithms, iterative improvement heuristics and the Kernighan-Lin procedure is established.

Keywords: integer programming, upper bounds, test sets, Buchberger algorithm, Gröbner bases, iterative improvement heuristics.

1 Introduction

In this paper we consider an integer programming problem of the type

$$\text{(IP)} \qquad \max \{c^T x : \quad Ax \leq b, \quad 0 \leq x \leq u, \quad x \text{ integral}\},$$

where $A \in \mathbb{N}^{m \times n}$ is a given matrix, $b \in \mathbb{N}^m$ is the right hand side vector, $u \in \mathbb{N}^n$ denotes a vector of upper bounds for the variables and $c \in \mathbb{Z}^n$ is the objective function. (Here and in the following $\mathbb{N}$ denotes the natural numbers including 0.) We note at this point that without loss of generality we may assume $c_i > 0$ for all i: otherwise we can set $x_i = 0$ for the corresponding variable and eliminate a column from our program. If upper bounds are not explicitly given, they may be generated by setting $u_i := \min\{b_j/a_{ij} : a_{ij} > 0\}$. The algorithms and proofs that we present can easily be adapted to the solution of families of integer programs with varying right hand sides b, as long as finite upper bounds for the variables are given.

Integer programs (IP) can, in principle, be solved by applying the Buchberger algorithm [4] for computing the Gröbner basis of a toric ideal. The connection between test sets for integer programming and Gröbner bases of certain ideals was first established by Conti & Traverso [5], see also Thomas [16]. Whereas the algorithms of [5] and [16] deal with families of integer programming problems of the form $Ax = b$, $x \geq 0$ for varying right hand side vectors b, we here show how to handle the case of a fixed right hand side and fixed upper bounds on the variables.

*Address after January 1, 1995: Department of Mathematics 6–1, Technical University of Berlin, 10623 Berlin, Germany, ziegler@math.tu-berlin.de

This is essential, since most integer programs arising "in practice" have upper bounds, often $u_i = 1$.

Moreover, the procedures formulated in [5] and [16] are applied to an "extended" integer program with additional variables, of the form

$$(EIP(b)) \qquad \min \{c^T x + M\mathbf{1}^T y : \quad Ax + Ey = b,\ x \in \mathbb{N}^n,\ y \in \mathbb{N}^m,\ x, y \text{ integral}\},$$

where $M \in \mathbb{N}$ is a "large" integer, E is the $m \times m$ identity matrix, and $\mathbf{1}$ denotes the vector of all ones. In practice the additional variables may lead to considerable increase in the space and time requirements of the algorithms considered. The original proofs for correctness and finiteness of those algorithms needed an algebraic machinery (which only applies in the case $c \geq 0$); however, the geometric version by Thomas [16] only needed the Gordan-Dickson lemma.

We formulate our algorithm in the original space and interpret all steps geometrically. In fact, three variants of a (simple) algorithm are presented, and we give elementary geometric proofs for their correctness. A relationship between these (exact) algorithms, iterative improvement heuristics and the Kernighan-Lin procedure is established as well. Finally, preliminary computational results show that this type of algorithms has a potential for becoming a powerful tool in the solution of practical integer programming problems.

As mentioned above, our variant of the Buchberger algorithm deals with an integer programming problem of the type

$$\max \{c^T x : \quad Ax \leq b, \quad 0 \leq x \leq u, \quad x \text{ integral}\}. \tag{1}$$

In order to handle more general programs

$$\max \{c^T x : \quad Ax \leq b, \quad Cx = d, \quad 0 \leq x \leq u, \quad x \text{ integral}\}, \tag{2}$$

we apply the following simple transformation. We define $c' := c + M\mathbf{1}^T C$ (where $\mathbf{1}$ is the vector with all components equal to 1 and M is a large integer), and solve the integer programming problem

$$\max \{c'^T x : \quad Ax \leq b, \quad Cx \leq d, \quad 0 \leq x \leq u, \quad x \text{ integral}\}. \tag{3}$$

Then every optimal solution x^0 of (3) will satisfy $Cx^0 = d$, provided that the program (2) is feasible. If (2) is infeasible, then the objective function value of an optimal solution to (3) is less than $M\mathbf{1}^T d$. Therefore, in terms of optimal solutions both formulations (2) and (3) are in a one-to-one correspondence, and we can always assume that the integer programming problem is given in the form (1).

Throughout the paper we use the following notation. N denotes the set $\{1, \ldots, n\}$. We say that $x \leq y$ holds for vectors $x, y \in \mathbb{Z}^n$ if $x_i \leq y_i$ for all $i \in N$. Thus "$\leq$" is a partial order on $\mathbb{Z}^n$.

From the objective function c we obtain a linear order on $\mathbb{Z}^n$ as follows: we choose an arbitrary term order $\prec_0$ (for example, lexicographic), and use it as a "tie breaker" on the points that have the same objective function value under c; that is, we define

$$x \ \prec_c \ y \qquad :\Longleftrightarrow \qquad \begin{cases} c^T x < c^T y, & \text{or} \\ c^T x = c^T y & \text{and } x \prec_0 y. \end{cases}$$

In the following "$\prec$" always denotes a linear order $\prec_c$ that refines the (fixed) objective function c in this way. One might note that $\prec$ is a term order in the sense of Gröbner basis theory if and only if $c \geq 0$. (In case of doubt we write "$\prec_c$" for "$\prec$.")

For a vector $d \in \mathbb{Z}^n$ we define $d^{\succ} := d$ if $d \succ 0$; in case that $d \prec 0$, we set $d^{\succ} := -d$. For $v \in \mathbb{Z}^d$ we denote by v^+ the vector with $v_i^+ = v_i$ if $v_i \geq 0$ and $v_i^+ = 0$, otherwise. Accordingly, v^- is the vector with $v_i^- = -v_i$ if $v_i \leq 0$ and $v_i^- = 0$, otherwise. Clearly $v = v^+ - v^-$.

Definition 1.1 Given a matrix $A \in \mathbb{N}^{m \times n}$, objective function coefficients $c_i \in \mathbb{Z}$, $i \in N$ and a right hand side vector $b \in \mathbb{N}^m$, we denote by $IP_{A,b,c,u}$ the optimization problem

$$(IP_{A,b,c,u}) \qquad \max\{c^T x : \quad Ax \leq b, \quad 0 \leq x \leq u, \quad x \text{ integral}\}.$$

We say that x is *feasible* for the integer programming problem $IP_{A,b,c,u}$ if $Ax \leq b$, $0 \leq x \leq u$ and x is integral.

A subset $B \subseteq \mathbb{Z}^d$ is a *test set* for the integer program $IP_{A,b,c,u}$ if and only if $-b \leq Av \leq b$, $-u \leq v \leq u$, $v \succ 0$ for all $v \in B$, and if for every non-optimal point $x \in \mathbb{N}^n$, there is some $v \in B$ such that $x + v$ is feasible.

The paper is organized as follows. In Section 2 we give an brief sketch of the approach of Conti & Traverso [5] and Thomas [16]. Section 3 presents the new variants of the Buchberger algorithm to compute test sets for integer programs. A link of these variants to iterative improvement heuristics and to the Kernighan-Lin heuristic is established in Section 4. We also show preliminary computational results when our algorithms are applied to small and medium sized real world problems in Section 5.

2 The Buchberger algorithm

In this section, we review the "Buchberger algorithm for integer programming" of Conti & Traverso [5]. Our presentation follows Thomas [16]. See Cox, Little & O'Shea [6] for basics on Gröbner bases and Buchberger's algorithm in the original setting (ideals in commutative rings) that motivated the constructions, and Hosten & Sturmfels for a recent report about computational results [11].

For the following exposition, we consider the integer programs for which $A \in \mathbb{N}^{m \times n}$ is a fixed, non-negative integer matrix. The right hand side vector, $b \in \mathbb{N}^m$, is considered as variable. Let $c \geq 0$ be a (fixed) linear objective function. We will also assume that the linear program

$$(LP(b)) \qquad \min c^T x : \quad Ax = b, \; x \geq 0$$

is bounded for every b. This is not much of a restriction: for example, it is satisfied if $c \geq 0$. In particular, this implies that the integer programs

$$(IP(b)) \qquad \min c^T x : \quad Ax = b, \; x \in \mathbb{N}^n$$

are bounded. With respect to the refined objective function $\prec$, we get that $IP(b)$ either has no feasible solution, or it has a unique optimal solution. We use $IP_{A,c}$ to denote the whole family of these integer programs, with fixed A and c, but varying right hand side b. The key idea is to consider this whole class of programs simultaneously. The following is immediate from Dickson's lemma [6, p. 70].

Lemma 2.1 *There is a* unique *minimal (finite!) set of vectors* $\{a_1, \ldots, a_t\} \subseteq \mathbb{N}^n$ *such that*

$$\{x \in \mathbb{N}^n : x \textit{ non-optimal}\} \quad = \quad \{x \in \mathbb{N}^n : x \geq a_i \textit{ for some } i\}.$$

Definition 2.2 A subset $B \subseteq \mathbb{Z}^d$ is a *test set* for the family $IP_{A,c}$ of integer programs if

- $Ag = 0$ and $g \succ 0$ for all $g \in B$, and
- for every non-optimal point $x \in \mathbb{N}^n$, there is some $g \in B$ with $x - g \geq 0$.

The definition of a "test set" immediately provides the following algorithm for integer programming — provided we have a feasible point to start with, and we know how to compute a test set.

Algorithm 2.3 to solve programs in a family $IP_{A,c}$:

(1) Input B, some $x \in \mathbb{N}^n$ (feasible for $IP(Ax)$).

(2) While x non-optimal

(2.1) find $g \in B$ such that $x - g \geq 0$.

(2.2) $x \longrightarrow x - g$.

Theorem 2.4 (Thomas [16, Cor. 2.1.10])
The unique minimal test set for the family $IP_{A,c}$ of integer programs is

$$B_c \;=\; \Big\{a_i - b_i : a_i \in \mathrm{MIN}_{\leq}\{x \in \mathbb{N}^n \text{ non-optimal}\},\ Aa_i = Ab_i,\ b_i \text{ optimal}\Big\},$$

where the symbol $\mathrm{MIN}_{\leq}$ denotes the set of minimal elements with respect to the partial order $\leq$.

The key observation is an identification between this "geometric" test set and an algebraic gadget: the minimal test set corresponds to a Gröbner basis of the "toric ideal"

$$I_A \;:=\; \langle x^{a^+} - x^{a^-} : Aa = 0,\ a \in \mathbb{Z}^n\rangle$$

with respect to the term order $\prec$.

The information that B_c is a Gröbner basis of I_A is by itself not too helpful, since in general one does not know a generating set for the ideal — so one cannot compute a Gröbner basis, either. Thus Conti & Traverso [5] create a larger integer program, which has an obvious integer feasible point, and for which the ideal has a nice generating set to start from. However, this leads to a problem of higher dimension, which tends to be computationally more difficult.

Conti & Traverso's "extended integer programs" are given by

$$(EIP(b)) \qquad \min_{\prec_{(M1,c)}} :\quad Ey + Ax = b,\ x \in \mathbb{N}^n, y \in \mathbb{N}^m$$

where $M \in \mathbb{N}$ is a large integer, E is the $m \times m$ identity matrix, and $\mathbf{1}$ denotes the vector of all ones. We use $EIP_{A,c}$ to denote the whole family of these integer programs, with fixed A and c, but varying right hand side b. What have we gained? On the one hand, all of the programs $EIP(b)$ are feasible: they have the obvious solutions $x = 0$, $y = b$. However, an optimal solution will satisfy $y = 0$, $x = x_0$ if the program $IP(b)$ is feasible, because M is sufficiently large. If $IP(b)$ is infeasible, then the extended program $EIP(b)$ has an optimal solution with $y > 0$. Putting both cases together, we see that it is sufficient to solve the extended programs.

Proposition 2.5 (Conti & Traverso [5]) *The ideal*

$$I_{(E,A)} \;:=\; \langle y^{a_1^+}x^{a_2^+} \;-\; y^{a_1^-}x^{a_2^-} :\ (E,A)\begin{pmatrix}a_1\\a_2\end{pmatrix} = 0,\ \begin{pmatrix}a_1\\a_2\end{pmatrix} \in \mathbb{Z}^{m+n}\rangle$$

is generated by the binomials

$$y^{Ae_j} - x_j \qquad \text{for } 1 \leq j \leq n.$$

The reduced Gröbner basis of $I_{(E,A)}$ with respect to the term order $\prec_{(M\mathbf{1},c)}$ yields the minimal test set $B_{(M\mathbf{1},c)}$ for that family $EIP_{A,c}$, via the canonical bijection

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \quad \longleftrightarrow \quad y^{a_1^+} x^{a_2^+} \; - \; y^{a_1^-} x^{a_2^-} .$$

Proof. The binomials $y^{Ae_j} - x_i$ form a Gröbner basis for $I_{(E,A)}$, for any lexicographic term order with $x_i \succ_{lex} y_j$. Thus they generate the ideal. □

Putting things together, we have a simple algorithm for integer programming: we "only" need to compute a reduced Gröbner basis with respect to the term order $\prec_{(M\mathbf{1},c)}$, and then use this with the above algorithm to solve the programs $IP_{A,c}$.

Algorithm 2.6 "Integer programming via Buchberger's algorithm" [5]
The following procedure solves the integer program

$$(IP(b)) \qquad \min_{\prec_c} : \quad Ax = b, \; x \in \mathbb{N}^n$$

for $A \in \mathbb{N}^{m \times n}$, $b \in \mathbb{N}^m$, $c \in \mathbb{N}^n$.

(1) **Compute a test set**

- (1.1) Input A, c.
- (1.2) Compute the reduced Gröbner basis $B_{(M\mathbf{1},c)}$ for $I := \langle y^{Ae_j} - x_j \rangle$.
- (1.3) Output the test set $B_{(M\mathbf{1},c)}$.

(2) **Optimization**

- (2.1) Input $B_{(M\mathbf{1},c)}$, b.
- (2.2) Reduce y^b with respect to $B_{(M\mathbf{1},c)}$, to get $y^{a_1} x^{a_2}$.
- (2.3) if $a_1 \neq 0$, output "$IP(b)$ is infeasible"
  if $a_1 = 0$, output "$x_0 = a_2$ is optimal."

While the first phase of this algorithm is hard work, the second one is quite trivial (if we manage to efficiently search the Gröbner basis, which may be huge). If we don't have a complete Gröbner basis, then we can still use any partial basis to reduce the monomial y^b, which may yield a feasible, or even the optimal, point.

The Buchberger algorithm for integer programming is just a special case of the general Buchberger algorithm. However, there is a lot of special features in the situation of "toric ideals" we consider here. In particular, one only has to deal with "binomials with disjoint supports": thus we can get an entirely geometric formulation of the algorithm, dealing with lattice vectors in $\mathbb{Z}^n$ — no polynomials whatsoever appear. This simplifies the data structures considerably.

The first observation is that one is dealing with

$$I := \langle y^{Ae_j} - x_j \rangle,$$

a binomial ideal. (See [7] for more on binomial ideals.) The S-pair of any two binomials is a binomial. Also the reduction of a binomial by binomials yields a binomial. Thus the reduced Gröbner basis of I consists of binomials.

As a second step, notice that in this situation, if $x^a - x^b \in I$ (with $a, b \geq 0$) is a binomial such that the two monomials have common factors, then we can remove them: $x^{(a-b)^+} - x^{(b-a)^+}$ is also contained in I. Let

$$\mathcal{L} \;:=\; \mathbb{Z}^{m+n} \,\cap\, \{\begin{pmatrix} y \\ x \end{pmatrix} : Ey + Ax = 0\}.$$

Then the reduced Gröbner basis of I consists of binomials of the form $x^{a^+} - x^{a^-}$ with $a \in \mathcal{L}$. Thus we are really dealing with a geometric algorithm operating on lattice vectors. This is worked out very nicely in Thomas [16]. In particular, one can translate the complete procedure for computing the (unique!) reduced Gröbner basis into this geometric setting.

In this vein, the following two geometric algorithms compute *the reduced Gröbner basis of a lattice*, that is, the finite subset $B \subseteq \mathcal{L}^{\succ 0}$ that corresponds to the reduced Gröbner basis of $I_{\mathcal{L}}$. The assumption for this is that we know a "good" generating set for the lattice, i.e., a subset of the lattice corresponding to a set of binomials that generates $I_{\mathcal{L}}$.

Algorithm 2.7 "Reduction"
The following algorithm computes the reduction of a vector $f \in \mathbb{Z}^{m+n}$ by a set B of integer vectors.

(1) Input $B \subseteq \mathcal{L}^{\succ 0}$, $f \succ 0$.

(2) As long as possible perform the following steps:

(2.1) If there is some $g \in B$ with $g^+ \leq f^+$, then replace f by $(f - g)^{\succ}$.

(2.2) If there is some $g \in B$ with $g^+ \leq f^-$, then replace f by $f + g$.

(3) Output $\overline{f} := f$.

Algorithm 2.8 "Buchberger algorithm on lattice vectors" [16]
The following algorithm computes the reduced Gröbner basis of the lattice $\mathcal{L}$, for a fixed term order $\succ$.

(1) **Construct a Gröbner basis**

(1.1) Input a basis $\{a_1, \ldots, a_n\} \subseteq \mathcal{L}$ of the lattice $\mathcal{L}$ such that the binomials $x^{a^+} - x^{a^-}$ generate $I_{\mathcal{L}}$.

(1.2) Set $B_{old} := \emptyset$, $B := \{a_1, \ldots, a_n\}$

(1.3) While $B_{old} \neq B$, repeat the following steps

(1.3.1) $B_{old} := B$

(1.3.2) (S-pairs) construct the pairs $f := a - a' \succ 0$ with $a, a' \in B$.

(1.3.3) (Reduction) reduce the vectors f by the vectors in B_{old}, obtain $\overline{f}$,

(1.3.4) if $\overline{f} \neq 0$, set $B := B \cup \overline{f}$.

(2) **Construct a minimal Gröbner basis**

(2.1) As long as possible perform the following step:
If for some $g \in B$ the point g^+ can be reduced by some $g' \in B \backslash g$, then delete g from B.

(3) **Construct the reduced Gröbner basis**

(3.1) As long as possible perform the following step:
If for some $g \in B$ the point g^- can be reduced by some $g' \in B \backslash g$, then replace g by the corresponding reduced vector: $B := B \backslash g \cup \overline{g}$.

(3.2) Output $B_{red} := B$.

For a successful implementation it is important to reduce earlier, otherwise the Gröbner bases as constructed in the "First Step" will be too large. See Conti & Traverso [5] and Moulinet & Pottier [14] for further ideas about how to make this efficient.

3 Three variants of the Buchberger algorithm

In this section we present three variants of an algorithm that computes a test set for the integer programming problem

$$(IP_{A,b,c,u}) \qquad \max\{c^T x : Ax \leq b,\ x_i \in \{0, 1, \ldots, u_i\},\ i \in N\}$$

where $A \in \mathbb{N}^{m \times n}$, $c \in \mathbb{N}^n$, $b \in \mathbb{N}^m$ is a *fixed* right hand side vector, and u is the vector of upper bounds on the variables. In the following $\prec$ always denotes a term order refining c.

We start with an outline of the basic form of the algorithm. Having proved that this version of the algorithm terminates after finitely many steps with a test set for $IP_{A,b,c,u}$, we show how to speed up the computations by excluding certain vectors in the computation of the test set.

Roughly speaking, a test set B can be computed as follows. Start with the n unit vectors, $B := \{e_i :\ i \in N\}$. Iteratively, compute the difference vectors between all pairs of vectors in B and direct each such difference vector such that it is greater than 0 with respect to the order. All such difference vectors that are not in B and that are differences of feasible vectors for $IP_{A,b,c,u}$ are added to B. The algorithm terminates if no more vectors are added to B.

Algorithm 3.1

(1) Set $B_{old} := \emptyset$, $B := \{e_i :\ i \in N\}$.

(2) While $B_{old} \neq B$ perform the following steps:

(2.1) Set $B_{old} := B$.

(2.2) For all pairs of vectors $v, v' \in B$ with $v \prec v'$ that satisfy $-b \leq A(v' - v) \leq b$ and $-u \leq v' - v \leq u$, set $B := B \cup \{v' - v\}$.

Whenever Step (2) of this algorithm is executed (except for the last time), a new vector is added to the set B. Since the number of different vectors $w = v' - v$ satisfying $-u \leq w \leq u$ is bounded by $\prod_{i \in N}(2u_i + 1)$, the above algorithm terminates after finitely many steps.

We now show that the set B generated by Algorithm 3.1 is a test set for $IP_{A,b,c,u}$. Suppose that x is a feasible point ($Ax \leq b$, $0 \leq x \leq u$) that is not optimal and that cannot be improved by adding an element in B. Let x' be some feasible vector with $x' \succ x$. Then $x' - x$ can be written as an integral combination of unit vectors: we can decrease from x to reach 0 (staying feasible), then increase to reach x', using only vectors in B. Hence, there is a sequence $P = (x^0, \ldots, x^p)$ of vectors x^i with the following properties:

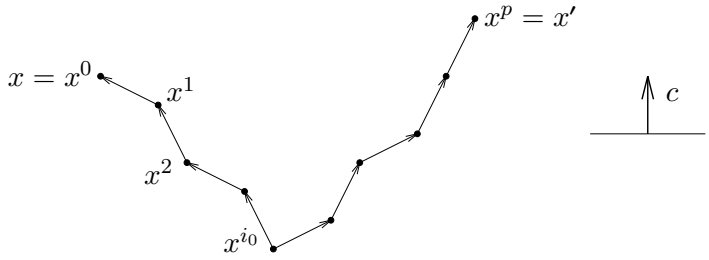
(i) $x^0 = x$, $x^p = x'$,

(ii) for all $i = 1, \ldots, p$, $(x^i - x^{i-1})^\succ \in B$, and

(iii) all the points $x^i \in P$ are feasible.

In fact, in the specific current situation, we know more for (ii): there is some i_0 such that $-(x^i - x^{i-1}) \in B$ for $i \leq i_0$, and $(x^i - x^{i-1}) \in B$ for $i > i_0$.

In the following sketches, the vertical direction "upwards" represents increasing objective function. Thus the small vectors, depicting elements of B, are directed upwards.



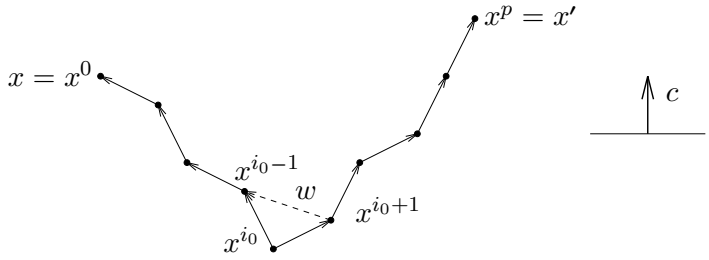
Among all sequences that satisfy (i), (ii) and (iii), let $P = (x^0, \ldots, x^p)$ be a sequence such that the minimum point in P, x^{i_0}, is maximal with respect to the order $\prec$. (Such a sequence exists since the number of feasible points is finite. The minimum point x^{i_0} is unique since $\succ$ is a total order.)

We have $i_0 \neq 0$ (otherwise $x = x^0$ could be improved by $x^1 - x^0 \in B$), and $i_0 \neq p$ (otherwise we would have $x' = x^p \prec x^0 = x$).

Both vectors x^{i_0+1} and x^{i_0-1} are feasible. It follows that

$$w \;:=\; \Big((x^{i_0+1} - x^{i_0}) \;-\; (x^{i_0-1} - x^{i_0})\Big)^\succ \;=\; (x^{i_0+1} - x^{i_0-1})^\succ$$

satisfies $-b \leq Aw \leq b$ and $-u \leq w \leq u$. Moreover, since $x^{i_0-1} - x^{i_0} \in B$ and $(x^{i_0+1} - x^{i_0}) \in B$, the difference vector w has been computed in Step 2.2 of Algorithm 3.1, and was added to B.



Thus

$$P' \;:=\; (x^0, \ldots, x^{i_0-1}, x^{i_0+1}, \ldots, x^p)$$

again satisfies properties (i) to (iii), yet the minimum element in P' is larger than x^{i_0}: a contradiction. □

With this we have proved the following theorem.

Theorem 3.2 *Algorithm 3.1 terminates after a finite number of steps. The output is a test set for the integer programming problem $IP_{A,b,c,u}$.*

Example 3.3 Consider the 0/1 knapsack problem

$$\max\{x_1 + 2x_2 + 3x_3 :\; x_1 + 2x_2 + 3x_3 \leq 3,\; x_i \in \{0, 1\},\; i = 1, 2, 3\}.$$

The algorithm starts with the vectors e_1, e_2, e_3. Then the vectors $e_2 - e_1$, $e_3 - e_1$ and $e_3 - e_2$ are added to the set B. In the next iteration the vectors $e_3 + e_1 - e_2$ and $e_1 + e_2 - e_3$ are added. The algorithm terminates with the set

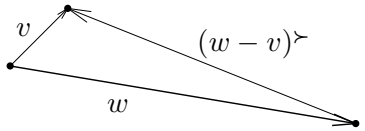
$$B \;=\; \{e_1, e_2, e_3, e_2 - e_1, e_3 - e_1, e_3 - e_2, e_3 + e_1 - e_2, e_1 + e_2 - e_3\},$$

since in Step (2) no new vectors are found.

This set is, indeed, a test set for the above 0/1 knapsack problem. Yet, not all of these vectors are really needed to guarantee that one can go from any feasible point of this program to the optimal solution without decreasing the objective function value in each step. Namely, the vector $e_3 + e_1 - e_2$ can always be replaced by the two vectors e_1 and $e_3 - e_2$, both being elements in B. In this situation we call $e_3 + e_1 - e_2$ *reducible*. Elimination of reducible vectors from the computation of a test set is a very important issue in order to keep the size of the test set small. We now formalize this situation.

Definition 3.4 For an integer programming problem $IP_{A,b,c,u}$, let B be a family of improvement vectors (that is, all $v \in B$ satisfy $-u \le v \le u$ and $-b \le Av \le b$, as well as $v \succ 0$).

A vector $w \neq 0$ *can be reduced by* $v \in B$ if $v^+ \le w^+$, $v^- \le w^-$, and $(Av)^+ \le (Aw)^+$. In this situation, we say that we obtain $(w - v)^{\succ}$ by *reducing* w.



In this situation, a trivial computation shows that if at any feasible point $x \in \mathbb{Z}^n$ the vector w can be applied (that is, if $x + w$ is feasible as well), then one could also apply v instead of w, and obtain a feasible point $x + v \succ x$ (and after that one can apply $w - v$ to reach $x + w$). Thus we note:

- if x and $x + w$ are feasible, then so is $x + v$,
- $|v|_1 \le |w|_1$, with equality only if $v = w$, and $|w - v|_1 < |w|_1$, and
- $x + v \succ x$.

We have *not* assumed that $w \succ 0$, because in the following proofs of this section we need to reduce difference vectors of the form $w - w'$.

Algorithm 3.5 "Reduction"
This algorithm computes the *reduction* $\overline{w}^B$ of a vector $w \in \mathbb{Z}^n$ by a set B of improvement vectors.

(1) Input $B \subseteq (\mathbb{Z}^n)^{\succ 0}$, $w \in \mathbb{Z}^n$.

(2) As long as possible, find $v \in B$ such that $r \in \{w, -w\}$ can be reduced by v, and replace w by $r - v$.

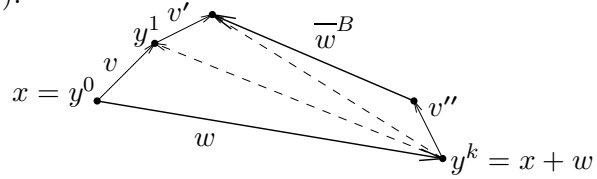
(3) Output $\overline{w}^B := r^{\succ}$.

The vector $\overline{w}^B$ is called the *reduced vector* of w with respect to B.

Proposition 3.6 *Assume that x and $x+w$ are feasible, and compute $\overline{w}^B$. Then there is a sequence of distinct integral points $x = y^0, y^1, \ldots, y^{k-1}, y^k = x+w$, with the following properties:*

- *each y^j is feasible,*
- *$y^0 \prec y^1 \prec \ldots \prec y^{j_0} \succ \ldots \succ y^k = x+w$, for some $0 \le j_0 \le k$,*
- *in particular, for $0 < j < k$ we get $y^j \succ x$, or $y^j \succ x+w$, or both,*
- *$(y^j - y^{j-1})^\succ \in B$ for each $1 \le j \le k$, except that if $\overline{w}^B \neq 0$, then we either have $y^{j_0} - y^{j_0-1} = \overline{w}^B$, or $y^{j_0} - y^{j_0+1} = \overline{w}^B$, and*
- *$|y^j - y^{j-1}|_1 \le |y^k - y^0|_1 = |w|_1$, with equality only if $k = 1$.*

The following sketch shows how this sequence of points y^j may be generated, by reducing by v, v' and v'' (in that order).



Using reduction, we can modify our initial algorithm as follows.

Algorithm 3.7

(1) Set $B_{old} := \emptyset$, $B := \{e_i : i \in N\}$.

(2) While $B_{old} \neq B$ repeat the following:

(2.1) Set $B_{old} := B$.

(2.2) For all pairs of vectors $v, v' \in B_{old}$ such that $v \prec v'$ perform the following steps:

(2.2.1) If $-b \le A(v'-v) \le b$ and $-u \le v'-v \le u$, set $w := v'-v$.

(2.2.2) Compute $r := \overline{w}^B$ by Algorithm 3.5.

(2.2.3) Set $B := B \cup \{r\}$.

Again, Algorithm 3.7 terminates after finitely many steps, since there exists an upper bound on the number of different vectors that can be added to the set B.

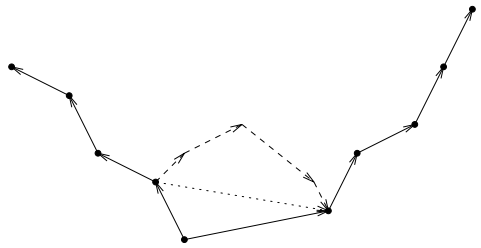
To show that it computes a test set, we can nearly proceed as before. For any non-optimal feasible point x that cannot be improved by a vector in B, and every feasible $x' \succ x$, there is a sequence $P = (x^0, \ldots, x^p)$ that satisfies the properties (i) to (iii) above, for which the minimum point x^{i_0} occuring in P is maximal (with respect to $\prec$). We again have $0 < i_0 < p$, which means that the difference vector

$$w \;=\; (x^{i_0+1} - x^{i_0-1})^\succ$$

was considered by Algorithm 3.7. Thus, by Proposition 3.6 we get a new sequence of feasible points

$$P' \;:=\; (x^0, \ldots, x^{i_0-1} = y^0, y^1, \ldots, y^k = x^{i_0+1}, \ldots, x^p),$$

where the minimum point of P' is larger than that of P. □



Thus we have proved the following.

Theorem 3.8 *Algorithm 3.7 terminates after a finite number of steps. The output is a test set for the integer programming problem $IP_{A,b,c,u}$.*

As a postprocessing step, the size of the final output, B, of Algorithm 3.7 can be further reduced by the following algorithm.

Theorem 3.9 "Post-Processing"
Let B be any test set for $IP_{A,b,c,u}$. Then one can, successively for each $w \in B$, do

- *if w is reducible by some $v \in B\backslash w$, replace B by $B\backslash w$, and*
- *if $-w$ is reducible by some $v \in B\backslash w$, replace B by $(B\backslash w) \cup \{\overline{w}^{B\backslash w}\}$, otherwise.*

After these operations, B is still a test set.

Next we deal with the following question: what are sufficient conditions for a vector b to be successively reducible to 0? This question is of computational relevance, because vectors that can be reduced to 0 (by a sequence of reduction steps) need not be added to the set B during the run of Algorithm 3.7, but can be excluded in advance.

A first criterion in this respect can be adapted from Buchberger [4] (see also [6]). Suppose that $v, w \in B$ are two vectors in the current set B of Algorithm 3.7 with $v \succ w$. If the following three conditions are satisfied:

- the vectors v^+ and w^+ have disjoint support,
- the vectors v^- and w^- have disjoint support,
- the vectors $(Av)^+$ and $(Aw)^+$ have disjoint support,

then the vector $d = v - w$ (to be computed in Step 2.2 of Algorithm 3.7) is reducible by v and can be reduced to 0 (see step 2.2.2). This follows, because under the above assumptions we obtain:

$$d^+ = v^+ + w^-, \quad d^- = v^- + w^+, \quad \text{and} \quad (Ad)^+ = (Av)^+ + (Aw)^-.$$

Hence, every component in v^- is less or equal than the corresponding component of d^-; every component in v^+ is less or equal than the corresponding component of d^+ and every component in $(Av)^+$ is less or equal than the corresponding component of $(Ad)^+$. Therefore, d is reducible by v and since $r := v - d = w$ can certainly be reduced by w to 0, the statement follows.

Though such criteria help in reducing the running time of the overall procedure, the main bottleneck is that iteratively for every pair of vectors in the current set B the associated difference vector needs to be computed. Excluding parts of these computations in advance is one of the main issues for applying this algorithm to the solution of integer programming instances of nontrivial size.

We have found one such criterion. Namely we show that for every element in the current set B it is sufficient to compute just n difference vectors instead of $|B| - 1$. Then, however, the reduction step cannot be as strong as in Algorithm 3.7.

Algorithm 3.10

(1) Set $B_{old} := \emptyset$, $E := \{e_i : \ i \in N\}$, $B := E$.

(2) While $B_{old} \neq B$ repeat the following:

(2.1) Set $B_{old} := B$.

(2.2) For every $v := (w - e_i)^{\succ}$ with

(2.2.1) $w \in B$, $e_i \in E$ with $w \neq e_i$,

(2.2.2) $-b \leq Av \leq b$, $-u \leq v \leq u$, and

(2.2.3) v is not reducible by any $v' \in B$ with $v - v' \geq 0$

set $B := B \cup \{v\}$.

Theorem 3.11 *Algorithm 3.10 terminates after a finite number of steps. The output is a test set for $IP_{A,b,c,u}$.*

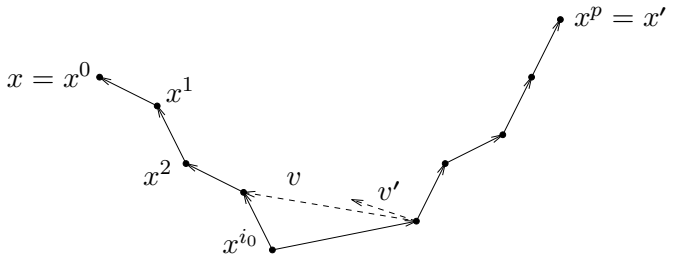
Proof. Finiteness of the algorithm is clear.

Suppose that there exists a non-optimal feasible point x that cannot be improved by any element of the set B that is computed by Algorithm 3.10. Let x' be the feasible vector with $x' \succ x$. As for Algorithm 3.1, there exists a sequence $P = (x^0, x^1, \ldots, x^{i_0}, \ldots, x^p)$ with $0 < i_0 < p$ and

(i) $x^0 = x$, $x^p = x'$,

(ii) $x^{i-1} - x^i \in E$ for $i \leq i_0$, and $x^{i+1} - x^i \in E$ for $i \geq i_0$,
except that one of $x^{i_0-1} - x^{i_0}$ and $x^{i_0+1} - x^{i_0}$ is permitted to be in $B \backslash E$.

(iii) every point x^i is feasible.



Now choose a sequence of points P with these properties such that its minimum point x^{i_0} is maximal.

In the course of Algorithm 3.10, one has computed the difference vector $v := (x^{i_0+1} - x^{i_0-1})^{\succ}$. This vector clearly satisfies (2.2.1) and (2.2.2). If it fails (2.2.3), then it can be written in the form $v = v' + \sum_{k=1}^{s} e_{i_k}$ for $v' \in B$ and $e_{i_1}, \ldots, e_{i_s} \in E$. Furthermore, we have that $x^{i_0 \pm 1} + v = x^{i_0 \mp 1}$ (where the sign "$\pm$" is "$+$" if $x^{i_0+1} \prec x^{i_0-1}$, and "$-$" otherwise), and thus all the points in the sequence

$$x^{i_0 \pm 1}, x^{i_0 \pm 1} + v', x^{i_0 \pm 1} + v' + e_{i_1}, \ldots, x^{i_0 \pm 1} + v' + \sum_{k=1}^{s} e_{i_k} = x^{i_0 \mp 1}$$

are feasible. From this we obtain a new sequence P' that again satisfies the above properties, yet the minimal point in this sequence is larger than x^{j_0}: a contradiction. □

It seems that after suitable post-processing, (cf. Algorithm 3.9, in analogy to the computation of the reduced Gröbner basis in Algorithm 2.8) the Algorithms 3.1, 3.7 and 3.10 produce the same (unique minimal?) test set. However, they proceed in different orders. While Algorithms 3.1 and 3.7 might produce exchange vectors of ℓ_1-norm 2^k in their k-th iteration of Step (2), Algorithm 3.10 will generate improvement vectors according to increasing ℓ_1-norm: it produces *all* improvement vectors of ℓ_1-norm k in the k-th iteration of Step (2).

4 A relation to iterative improvement heuristics

To simplify the discussions we will now only consider 0/1 problems, which have $u_i = 1$ for all $i \in N$. (Generalizations of what follows to arbitrary upper bounds are straightforward.) We furthermore assume that $Ae_i \leq b$ for all $i \in N$ and that for $i, j \in N$ the vectors Ae_i and Ae_j do not have disjoint support. These assumptions are usually satisfied by instances coming from travelling salesman problems, graph partitioning problems or knapsack problems etc.

One approach for obtaining good solutions for the problem

$$\begin{array}{lll} \max & \sum_{i=1}^{n} c_i x_i, & \\ & Ax \leq b & \text{for } j = 1, \ldots, m, \\ & x_i \in \{0, 1\} & \text{for } i = 1, \ldots, n, \end{array}$$

is to start with some feasible solution, i.e., a set $S \subseteq N$ such that $A\chi^S \leq b$. (For any subset $S \subseteq N$, χ^S denotes the incidence vector, with $\chi^S_i = 1$ if $i \in S$ and $\chi^S_i = 0$ otherwise.) Iteratively we replace items which belong to S by items which are not in the current solution via a certain rule such that the incidence vector of the resulting set, S' say, is feasible. Exchange the role of S with S' and repeat these steps until a certain stopping criterion is satisfied.

This procedure is certainly too general to be analyzed and needs specification of (a) the rule according to which items are replaced by others and (b) the stopping criterion.

In most implementations, exchange operations are allowed only if the number of items involved is less than or equal to a certain threshold value, λ say. More precisely, the cardinality of the symmetric difference between the sets S and S' must not exceed λ. The reason for that is simply to keep the running time of the procedure in acceptable limits. Indeed, usually a value of $\lambda = 2$ or $\lambda = 3$ is chosen. (The resulting algorithms are the "2-OPT" and "3-OPT" heuristics.) In addition, *iterative improvement heuristics* only allow exchanging items of S with items not in S if the objective function value $c\chi^S$ increases by this. Those algorithms terminate if the current solution x can not be improved by replacing items with $x_i = 1$ against items with $x_i = 0$ such that the number of items involved is less or equal than λ.

In case $\lambda = 2$ or $\lambda = 3$ there is a nice relationship between iterative improvement heuristics and our Algorithm 3.7. Similar statements can be made for Algorithms 3.1 and 3.10.

Proposition 4.1 *Let $v \in \{0, -1, 1\}^n$ be a vector such that $\sum_{i=1}^n |v_i| \leq 3$, $-b \leq Av \leq b$ and $v \succ 0$. After performing Step (2) in Algorithm 3.7 twice, the set B either contains v, or v is the sum of vectors in B.*

Proof. We start initially with the n unit vectors. When Step (2) is performed the first time all the vectors $e_i - e_j \succ 0$, $i, j \in \{1, \ldots, n\}$ are computed and added to B. Note that under the

assumptions introduced at the beginning of this section those vectors cannot be reduced. Thus, after a first processing of Step (2) all vectors $y \succ 0$ with entries $0, +1, -1$ and $\sum_{i=1}^{n} |y_i| \leq 2$, $-b \leq Ay \leq b$ have been generated.

Now let $v \in \{0,1\}^n$ be a 0/1 vector such that $\sum_{i=1}^{n} |v_i| = 3$, $-b \leq Av \leq b$ and $v \succ 0$. v can be written in one of the following forms: (i) $v = e_i - (e_u - e_w)$ with $(e_u - e_w) \succ 0$ or (ii) $x = (e_u - e_w) - e_i$ with $(e_u - e_w) \succ 0$ or (iii) $x = (e_u - e_w) + e_i$ with $(e_u - e_w) \succ 0$ or (iv) $v = e_u + e_w + e_i$ where $i, u, w \in \{1, \ldots, n\}$, $i \neq u \neq w \neq i$. Suppose that (iii) or (iv) holds. Then v is a sum of elements in B. Otherwise, (i) or (ii) is true. Then v is the difference vector of elements in B. This difference vector was computed by processing Step (2) of Algorithm 3.7 a second time. Since (iii) and (iv) are not true, this difference vector is not reducible via the elements in the current set B. □

As a corollary we obtain that via Algorithm 3.7 certain iterative improvement heuristics can be simulated. In fact, this algorithm is a strong generalization of the idea of iterative improvement heuristics and it is obvious that by restricting the number of times Step (2) is to be processed, the output can be used to (iteratively) improve feasible solutions.

Instead of admitting exchanges that always improve the current objective function value, Kernighan & Lin [12, 13] used a slightly different strategy. Again suppose that a set $S \subseteq N$ is given such that $A\chi^S \leq b$. Iteratively we either exchange one item which belongs to S by one item which does not so that the new solution is feasible again or we add to the current set S a new item if this yields a feasible solution. In other words, in order to move from a feasible solution x to a feasible solution x' we either have that $x + (e_i - e_j) = x'$ or $x + e_i = x'$ for some $i \in N$ (and $j \in N \setminus \{i\}$). Let $B^2 := \{e_i : \ i \in N\} \cup \bigcup_{e_i - e_j \succ 0} \{e_i - e_j\}$ and $B^2_- = \bigcup_{b \in B^2} \{-b\}$. Then, at a current feasible point x, Kernighan and Lin choose a vector v in $B^2 \cup B^2_-$ with $c^T v = \max\{c^T b : b \in B^2 \cup B^2_-, x + b \text{ is feasible}\}$. The procedure terminates if a given number of iterations have been performed.

Following this approach it is clear that at some point x an exchange operation may be performed that (locally) yields a decrease in the objective function. However by a sequence of exchanges, some of which might have a negative objective function value and some of which have a positive objective function value, we might reach some feasible point $x' \succ x$. Suppose, this is the case and in order to make our analysis easy let us also assume that x' can be reached from x by first applying an exchange step $v \in B^2_-$ and then an exchange step $w \in B^2$. We have already seen that the set B^2 is generated by performing Step (2) in Algorithm 3.7 once. Therefore, $-v \in B^2$ and $w \in B^2$ and as $x' \succ x$, so is $v + w \succ 0$. Since $v + w = w - (-v)$, with $(-v), w \in B^2$, the vector $x' - x$ is computed by performing Step (2) in Algorithm 3.7 a second time. Either, $x' - x$ is not reducible or it is. In the first case, it is added to our set B generated by Algorithm 3.7. In the latter case we can reach a point $\tilde{x}$ by using elements in B, such that in each step the objective function is not decreased.

Computing a difference vector w between pairs of elements in a current improvement set B, and directing it such that $w \succ 0$, can be viewed as a two step procedure, first locally getting worse, but afterwards globally improving the objective function value.

5 Computational results

In this section we present preliminary computational results with Algorithms 3.7 and 3.10. We have applied both algorithms to small and medium size instances coming from set covering problems (Steiner triple systems, see [15]), knapsack and multidimensional knapsack problems (see [9]), set partitioning problems [10] and experimental design problems (see [1]). For all

examples, except for those arising in experimental design problems, the number of columns is in the range of 6 to 105 and the number of rows is between 1 and 331. The set partitioning instances reported in [10] involve up to several thousands of columns and rows. From these original data we took subsets of the rows and columns and solved the set partitioning problem associated with this subset. For the instances of experimental design problems the number of columns varies from 147 to 2,205 and the number of rows is between 28 and 121.

We always start the computations with the vector 0, which is feasible for all instances (via the transformation of Section 1 we replace conditions $Ax = b$ by $Ax \leq b$). Iteratively we improve the current (feasible) solution by elements in the set B (computed according to Algorithms 3.7 and 3.10) until either we prove optimality or we exceed a time limit. We performed all tests on a SUN Sparc10 workstation with a limit of 45 minutes CPU time.

The tables below summarizes our results. In order to distinguish the instances we use the following convention: "knap" means that the instance is a (multidimensional) knapsack problem. The prefix "cov" and "part" stands for instances of set covering and set partitioning problems, respectively. The symbol is "des" used for experimental design problem instances. The first number behind the prefix corresponds to the number of columns. The second number is the number of rows. For example, knap.20.1 is an instance of a knapsack problem consisting of 20 items and 1 row, etc.

Column 2 of the tables gives the optimal solution of the corresponding problem. The optimal values for the set partitioning problems were obtained by the cutting plane code of [3]. For the knapsack problems this value was obtained with the cutting plane code reported in [8]. For the experimental design problem we refer to [1] for the optimal values. The optimal values for the set covering instances we took from MIPLIB [2]. Columns 3 and 4 report on the objective function value of the best solution found via Algorithm 3.7 and the corresponding time that was needed. Accordingly, columns 5 and 6 show the appropriate values if Algorithm 3.10 is applied.

The results show that in examples except for three the solution computed by Algorithm 3.10 is the same as the one given by Algorithm 3.7. In fact, both procedures nearly always behave the same concerning both running time and quality of the solution. It seems that none of the two variants is superior towards the other. Both procedures can prove optimality for the six instances cov.9.13, knap.6.10, knap.10.10, knap.20.1, part.10.4 and part.39.3 within 2 minutes of CPU time. For the remaining examples we did not succeed in proving optimality. Nevertheless for 45 out of 59 examples Algorithms 3.7 or 3.10 found an optimal solution.

For the set partitioning instances with more than 60 columns the algorithms perform rather poorly. In three cases the objective function values of the given solutions are still about 50 to 100% away from the optimal value and in two cases we do not even find a feasible set partitioning solution at all. For the six knapsack examples and the instance cov.45.331, where we did not find a solution with optimal value, the algorithms terminate with a feasible vector that is very close to the optimum in terms of objective function value. Finally, the algorithms behave quite specially on the experimental design instances: either the optimal solution is found immediately, or we do not find any feasible solution.

Summarizing our experiments, we think that on very hard combinatorial problems such as set partitioning we are still far from having a "real" optimization algorithm. It is not good enough to run Algorithm 3.10 as a "black box" that will hopefully find a good solution. Here, a *combinatorial* understanding of the test set B seems to be indispensible. For the knapsack, multidimensional knapsack and set covering problems that we tested, the situation is different. The Algorithms 3.7 and 3.10 work quite well on those instances and usually produce very good solutions. Moreover, the performance is stable. Certainly, the running times are still quite high and our implementation cannot in reasonable time handle instances with a couple of

EXAMPLE	OPT	SOL (3.7)	TIME (3.7)	SOL (3.10)	TIME (3.10)
cov.9.13	5	5	0:00	5	0:00
cov.15.36	9	9	0:00	9	0:00
cov.27.118	18	18	0:00	18	0:00
cov.45.331	30	29	0:01	29	0:01
knap.6.10	3800	3800	0:00	3800	0:00
knap.10.10	87061	87061	0:00	87061	0:00
knap.15.10	4015	4015	0:03	4015	0:03
knap.20.10	6120	6120	0:01	6120	0:01
knap.28.10	12400	12400	0:00	12400	0:00
knap.39.5	10618	10604	2:43	10604	2:40
knap.49.5	15223	15205	7:59	15205	7:53
knap.20.1	7708	7708	0:02	7708	0:02
knap.50.1	19928	19928	0:07	19928	0:07
knap.100.1	41773	41728	0:58	41728	0:58
knap.30.5	4561	4561	0:02	4561	0:02
knap.40.5	5557	5557	1:29	5557	1:29
knap.50.5	6159	6159	0:25	6159	0:25
knap.60.5	6954	6954	6:13	6954	6:16
knap.60.5	7486	7417	24:15	7486	33:01
knap.60.5	7289	7289	9:29	7289	9:31
knap.60.5	8633	8614	15:49	8614	15:50
knap.70.5	7698	7698	22:49	7698	22:48
knap.80.5	8947	8947	0:36	8947	0:36
knap.80.5	8344	8277	36:24	8277	35:57
knap.90.5	9492	9492	17:48	9492	17:51
knap.28.4	3418	3418	25:44	3405	9:01
knap.35.4	3186	3186	4:31	3186	4:31
knap.27.4	3090	3076	1:10	3076	1:10
knap.34.4	3186	3186	0:52	3186	0:52

EXAMPLE	OPT	SOL (3.7)	TIME (3.7)	SOL (3.10)	TIME (3.10)
knap.29.2	95168	95168	0:04	95168	0:04
knap.20.10	2139	2139	3:04	2122	0:03
knap.40.30	776	776	0:09	776	0:09
knap.37.30	1035	1035	9:23	1035	8:21
knap.28.2	130883	130883	1:11	130883	1:10
knap.105.2	624319	624319	1:39	624319	1:39
knap.60.30	7772	7772	0:26	7772	0:26
part.10.4	–4248	–4248	0:00	–4248	0:00
part.25.11	–7983	–7983	0:47	–7983	0:27
part.30.9	–1816	–1816	41:35	–1816	11:46
part.39.3	–2874	–2874	0:00	–2874	0:00
part.40.16	–8061	–8061	0:21	–8061	0:31
part.43.18	–11493	–11493	7:29	–11493	7:19
part.47.14	–7634	–7634	6:05	–7634	8:52
part.47.20	–6792	–6792	22:39	–6792	22:22
part.49.15	–22959	–22971	0:01	–22971	0:01
part.49.15	–5782	–5782	9:21	–5782	9:14
part.59.8	–2698	–3294	0:01	–3294	0:01
part.60.17	–11307	–22656	2:22	–22656	2:21
part.67.12	–5296	–8428	0:01	–8428	0:01
part.74.16	–11268	–15580	21:05	–15580	28:30
part.77.22	–16812	—	—	—	—
part.86.22	–9933	—	—	—	—
part.92.10	–2800	–2800	0:03	–2800	0:03
des.147.28	21	21	0:00	21	0:00
des.294.35	42	42	0:00	42	0:01
des.432.48	36	36	0:00	36	0:01
des.675.60	90	—	—	—	—
des.1014.91	78	78	0:03	78	0:02
des.2205.126	210	—	—	—	—

hundreds of columns. Storing and computing all the difference vectors would exceed the memory requirements.

To conclude, the algorithms that we presented here are very general, they are not adapted to special purpose problems and we implemented the routines straightforwardly. The quality of the solution that were produced on many of the test samples is quite high and very stable. We think that there is still a lot of research to be done in order to understand test sets combinatorially, but the results certainly indicate that the construction, analysis and adaption of such methods are worth further efforts.

Conclusions

Whereas dual methods like cutting plane algorithms have proven to be extremely successful in the solution of (large scale) integer programming problems, there is a lack of primal algorithms that have the potential to prove optimality of an integer program. In particular, it would be desirable to have both primal and dual algorithms that make it possible to systematically and simultaneously improve current primal and dual solutions. The three algorithms that we presented in this paper might have the potential to satisfy those needs and requirements.

Yet we are still far from applying our algorithms to large scale problem instances. From our point of view the computational results in Section 5 show that the "Buchberger type" algorithms generate very good solutions starting from scratch. Certainly the running time and the memory requirements form a bottleneck. The number of exchange vectors that are generated might even be squared when proceeding from one single iteration to the next. Hence, further research must concentrate on the combinatorial understanding of the exchange vectors that need to be contained in the test set. Then one could work with "classes of exchange vectors" implicitly rather than having to generate all such vectors explicitly. This would be analogous to the treatment of "classes of facets" in a cutting plane approach.

If one can make progress in this direction, then primal algorithms based on ideas as presented in this paper might become a powerful tool in the solution of integer programming problems.

Acknowledgements

This work was partially supported by Science Program SC1-CT91-620 of the EEC.
Günter M. Ziegler is supported by a "Gerhard-Hess-Forschungsförderpreis" of the German Science Foundation (DFG).

References

[1] Th. Beth, D. Jungnickel & H. Lenz: *Design Theory,* B.I.-Wissenschaftsverlag, Bibliographisches Institut, Zürich 1985; Cambridge University Press 1993.

[2] R. E. Bixby, E. A. Boyd & R. R. Indovina: *MIPLIB: a test set of mixed integer programming problems,* SIAM-News **25** (1992), p.16; available by electronically from Rice University SOFTLIB at `softlib.cs.rice.edu`, and from ZIB eLib electronic library at `elib.zib-berlin.de`.

[3] R. Borndörfer & R. Weismantel: *Solving set partitioning problems via cutting planes,* preprint ZIB-Berlin 1995, in preparation.

[4] B. Buchberger: *Gröbner bases: an algorithmic method in polynomial ideal theory,* in: N.K. Bose (ed.), "Multidimensional Systems Theory," D. Reidel 1985, 184-232.

[5] P. Conti & C. Traverso: *Buchberger Algorithm and Integer Programming,* Proceedings AAECC-9 (New Orleans), Springer LNCS **539**, 1991, pp. 130-139.

[6] D. A. Cox, J. B. Little & D. O'Shea: *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra,* Undergraduate Texts in Mathematics, Springer-Verlag, New York 1992.

[7] D. Eisenbud & B. Sturmfels: *Binomial ideals,* preprint 1994, 44 pages.

[8] C. E. Ferreira, A. Martin & R. Weismantel: *A cutting plane based algorithm for the multiple knapsack problem,* Preprint SC 93-7, ZIB-Berlin 1993, 29 pages.

[9] H. Heitkotter (Universität Dortmund): *Personal communication,* 1993.

[10] K. L. Hoffman & M. Padberg: *Solving airline crew-scheduling problems by branch-and-cut,* preprint 1992.

[11] S. Hosten & B. Sturmfels: *GRIN: An implementation of Gröbner bases for integer programming,* preprint 1994, 8 pages.

[12] B. W. Kernighan & S. Lin: *An efficient heuristic procedure for partitioning graphs,* Bell Systems Technical Journal **49** (1970), 291-307.

[13] B. W. Kernighan & S. Lin: *An effective heuristic algorithm for the traveling salesman problem,* Operations Research **21** (1973), 498-516.

[14] C. Moulinet & L. Pottier: *Gröbner bases of toric ideals: properties, algorithms, and applications,* preprint, INRIA Sophia Antipolis, 10 pages.

[15] A. Sassano: *On the facial structure of the set covering polytope,* Math. Programming **44** (1989), 181-202.

[16] R. R. Thomas: *A geometric Buchberger algorithm for integer programming,* Math. Operations Research, to appear.