OCTANE: A New Heuristic for Pure 0–1 Programs

Egon Balas [1], Sebastián Ceria [2], Milind Dawande [3]
Francois Margot [4]
Gábor Pataki [5]

January 1998
Revised May 1999

[1]GSIA, Carnegie Mellon University, Pittsburgh, PA 15213.
[2]Graduate School of Business, Columbia University, New York, NY 10027.
[3]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
[4]Department of Mathematical Sciences, Michigan Technological University, Houghton, MI 49931.
[5]Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027

Abstract

We propose a new heuristic for pure 0–1 programs, which finds feasible integer points by enumerating extended facets of the octahedron, the outer polar of the unit hypercube. We give efficient algorithms to carry out the enumeration, and explain how our heuristic can be embedded in a branch-and-cut framework. Finally, we present computational results on a set of pure 0–1 programs taken from MIPLIB and other sources.

1 Introduction

There is clearly a renewed interest in the research community in computational integer programming. The recent success of branch-and-cut as a solution framework for general integer programs, has revived an area that for a long time had fallen out of favor. Most of this success, however, has been obtained in the area of finding exact solutions for this class of problems, while very little effort has focused on the approximate solution of general 0–1 programs. In fact, the literature on heuristics for general integer programs is quite limited, with the main proposals to be found in no more than half a dozen references over the last 30 years [6, 13, 15, 16, 17, 19]. The only heuristic that has been extensively tested with results reported in the open literature is the pivot-and-complement procedure [6] developed two decades ago. Most of the effort in finding approximate solutions to hard 0–1 programs has focused on special classes of problems.

This lack of results, however, does not mean that practitioners and researchers are not interested in finding heuristic solutions to general 0–1 programs. On the contrary, it can be argued that most of the time, one cannot afford the computational effort required to get an optimal solution to a 0–1 program. This creates a hard problem for any practitioner interested in feasible solutions that may not necessarily be optimal; namely, how to get heuristic solutions for problems where so few heuristic procedures are available. Faced with this problem, most practitioners resort to what has become the method of choice: run a branch-and-bound or branch-and-cut algorithm with a time limit or until the first feasible integer solution has been found.

In this paper we propose a new heuristic, called OCTANE. It may be seen as one of the building blocks of the algorithm for constructing the so called enumerative intersection cuts [2, 12, 14]. Its basis is the one-to-one correspondence between $0-1$ points in $\mathbb{R}^n$ and the facets of the n-dimensional octahedron. From x, a fractional solution to the LP-relaxation of our $0-1$ program, OCTANE selects a direction a, then computes the first k facets of an octahedron containing x that are intersected by the half line originating at x and having direction a. Using the above-mentioned one-to-one correspondence, this yields k 0–1 points, used as potential solutions of the IP under consideration.

In Section 2 we introduce the necessary notation and give a more precise high-level description of OCTANE. Section 3 deals with the enumeration of the first k facets of the octahedron that are intersected by a given half line originating inside the hypercube. We first obtain a simple algorithm for finding the first facet intersected by the half line and then show how to perform an efficient enumeration of the first k intersected facets using the Reverse Search paradigm of Avis and Fukuda [1]. In Section 4, we modify the algorithm of Section 3 in order to generate extreme points of a truncated hypercube instead of the hypercube.

Section 5 discusses important implementation issues, in particular the choice of the endpoint x and direction a of the half line used for the enumeration and the number k of intersections to be computed. When embedding the heuristic inside a branch-and-cut procedure, several additional choices have to be made, for example deciding from which nodes of the branch-and-cut tree the heuristic should be called. These issues are covered in Section 5 too. Finally, in Section 6 we report computational results obtained on a set of test problems from MIPLIB [11].

2 Basic heuristic

Consider a $0-1$ integer program of the form:

$$\begin{array}{lll} \text{Min} & cx & \\ s.t. & Ax \geq b & (IP\ ) \\ & x_i \in \{0,1\},\ (i=1,\ldots,n) & \end{array}$$

We baptized our heuristic OCTANE, for OCTAhedral Neighborhood Enumeration. OCTANE performs a local search in the integer neighborhood of a fractional LP-solution of *(IP)*. The local search can be described as follows. Let K be the unit hypercube centered at the origin, and K^* the regular octahedron circumscribing it, i.e.:

$$\begin{aligned} K &= \{\, x \in I\!R^n \ : -\frac{e}{2} \leq x \leq \frac{e}{2} \,\} \\ K^* &= \{\, x \in I\!R^n \ : \|x\|_1 \leq \frac{1}{2}n \,\} \\ &= \{\, x \in I\!R^n \ : \delta x \leq \frac{1}{2}n,\ \forall\, \delta \in \{\pm 1\}^n \,\} \end{aligned}$$

where e is the vector of all ones. K^* is the *outer polar* of K as defined in [4], i.e. the polar of K scaled by a factor that causes K^* to circumscribe K. Each facet of K^* contains exactly one vertex of K, and vice versa, every vertex of K is contained in exactly one facet of K^*. Therefore there is also a one-to-one correspondence between the vertices x of $K + \frac{1}{2}e$, (i.e. all $0-1$ points) and facets δ of $K^* + \frac{1}{2}e$, given by

$$\frac{1}{2}\delta + \frac{1}{2}e \quad = \quad x$$

The heuristic works by computing the first k intersections of a half line originating at a fractional solution to the LP-relaxation of *(IP)* with the extended facets of $K^* + \frac{1}{2}e$, i.e. with the hyperplanes defining the facets. (In the sequel, we do not distinguish between facets and extended facets, or the normal δ to the inequality defining the facet). For notational convenience we shall carry out the enumeration in K^*, i.e. the octahedron centered at the origin. For some fixed value of the parameter k, the heuristic performs these steps:

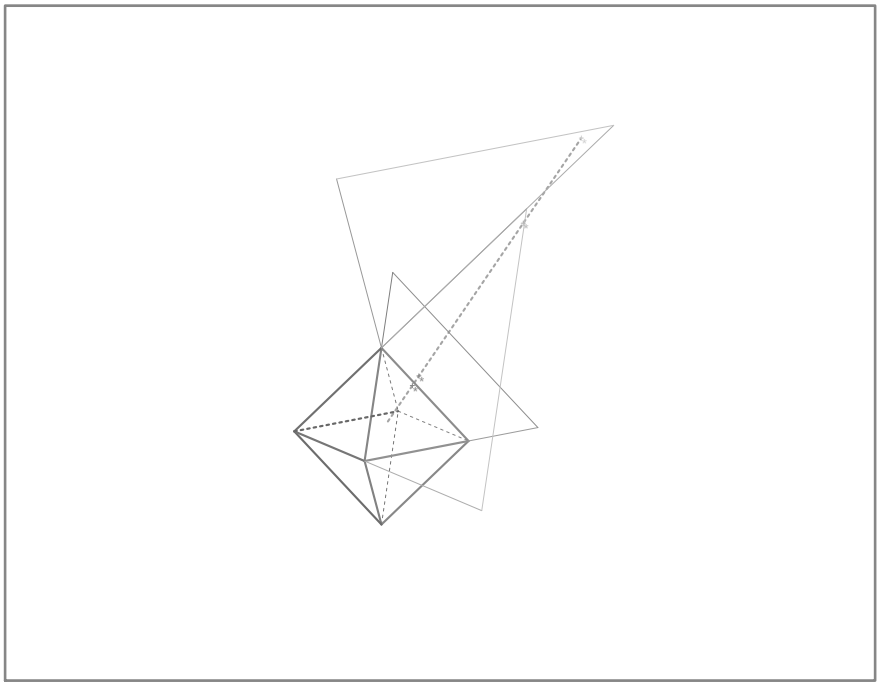

Figure 1: Intersection of the facets of the octahedron in Example 2.1

1. Let x be a fractional LP-solution of *(IP)* and $\bar{x} = x - \frac{1}{2}e$.

2. Choose a vector $a \in I\!R^n$ and consider the half line

$$r \quad = \quad \{\bar{x} + \lambda a \mid \lambda \geq 0\}$$

   originating at $\bar{x}$ with direction a.

3. Find $\{\delta^1, \ldots, \delta^k\}$, the first k facets of K^* intersected by r and the corresponding $0-1$ points $\mathcal{X} = \{x^1, \ldots, x^k\}$.

4. The points in $\mathcal{X}$ that are feasible for *(IP)* serve as heuristic solutions.

Notice that since x is fractional, $\bar{x} \in \text{int } K^*$.

Example 2.1 Consider the enumeration problem in $I\!R^3$ with

$$\begin{aligned} \bar{x} &= (-0.33, -0.36, -0.45) \\ a &= (0.1, 0.6, 0.9). \end{aligned}$$

Then the set of facets intersected by $r = \bar{x} + \lambda a$ and the corresponding values of λ are

$$\begin{array}{rcl} \delta^1 = (- + +) & , & \lambda = 1.41 \\ \delta^2 = (+ + +) & , & \lambda = 1.65 \\ \delta^3 = (+ - +) & , & \lambda = 4.8 \\ \delta^4 = (- - +) & , & \lambda = 6.3. \end{array}$$

Here we simply write $+$ for $+1$, and $-$ for -1, and we shall frequently use this notation in the rest of the paper as well. Observe that r intersects all "top" facets of K^* in clockwise order, as shown on Figure 1. □

A more advanced version of OCTANE uses the knowledge of the constraint set of *(IP)* at the enumeration stage, before checking feasibility. We construct a set of inequalities with disjoint supports and valid for all $0-1$ solutions of *(IP)*

$$z_0^i \leq s^i x \leq s_0^i \; (i = 1, \ldots, r)$$

and modify the enumeration algorithm to skip all the facets of K^* corresponding to $0-1$ points that are not feasible for these inequalities. This will be covered in Section 4.

The local search procedure described above may seem somewhat unnatural at first, thus it is worth giving a brief outline of its origin. The facet-enumeration problem first arose in the context of generating *enumerative intersection cuts* for *(IP)*. These cuts are based on the results sketched below.

Let $\tilde{x}$ be a nondegenerate fractional vertex (basic feasible solution) of the LP-relaxation of *(IP)* and $r_1, \ldots, r_n$ be the half lines originating at $\tilde{x}$ and pointing towards the adjacent vertices of the polyhedron. Let $\{\xi^{i1}, \xi^{i2}, \ldots, \xi^{ik}\}$ be the points in which r_i intersects the 1st, 2nd, ..., kth facet of $K^* + \frac{1}{2}e$ intersected by r_i in this order, and $x^{i1}, x^{i2}, \ldots, x^{ik}$ the $0-1$ points contained in those facets.

Theorem 2.2 *(1) Let $\alpha x = \beta$ be the unique hyperplane containing the n points $\xi^{11}, \ldots, \xi^{n1}$, with $\alpha\tilde{x} < \beta$. Then the inequality $\alpha x \geq \beta$, which cuts off $\tilde{x}$ is satisfied by all feasible 0–1 points.*

(2) Let $\alpha' x = \beta'$ be the unique hyperplane containing the n points $\xi^{1k_1}, \ldots, \xi^{nk_n}$ for some $k_1 \geq 1, \ldots, k_n \geq 1$, with $\alpha'\tilde{x} < \beta'$. Then the inequality $\alpha' x \geq \beta'$, which cuts off $\tilde{x}$ is satisfied by all feasible 0–1 points, with the possible exception of $x^{1,1}, \ldots, x^{1,k_1-1}, \ldots, x^{n,1}, \ldots, x^{n,k_n-1}$. □

The inequality $\alpha x \geq \beta$ of (1) is a special case of an *intersection cut*, which can be derived in the above fashion using any closed convex set in the role of $K^* + \frac{1}{2}e$ that contains $\tilde{x}$ but no $0-1$ point in its interior (Balas [2] used a sphere; $K^* + \frac{1}{2}e$ was first proposed

in Balas, Bowman, Glover and Sommer [5]). The paper [2] also treats the degenerate case. The inequality $\alpha' x \geq \beta'$ of (2) is a special case of an *enumerative-intersection cut*, an idea proposed by Burdet [12] and Glover [14]. While an inequality of this type does cut off $0-1$ points, these can be listed and checked for feasibility, hence the inequality can be added to the formulation of *(IP)* with no harm. The conceived strength of such a procedure is twofold: the enumeration makes the cut deeper, and may produce good heuristic solutions of *(IP)*, as the enumerated $0-1$ points are typically close to the fractional optimum $\tilde{x}$.

We started our research with the goal of testing enumerative intersection cuts. We found that whenever the enumerated $0-1$ points were feasible, they indeed turned out to be good heuristic solutions of *(IP)* . This led us to focus on the second aspect of the enumeration algorithm (i.e. using it as a heuristic). On the one hand, we experimented with different directions, starting points for the enumeration, and several other options. We also developed faster algorithms (both theoretically and practically) for the enumeration.

3 Enumerating the facets of the octahedron

3.1 A simple algorithm

In this section we describe a simple algorithm to enumerate the facets of the octahedron in the order that they are intersected by a given half line, starting from its endpoint $\bar{x}$. We first show that if δ is a facet intersected by the half line, but δ is not the facet intersected first, then by changing the sign of a single entry of δ, one can get a facet δ' intersected before δ. This was already observed by Balas [3] and yields a simple algorithm to find the facet intersected first.

Consider the octahedron

$$K^* \quad = \quad \{x \in I\!R^n \ : \delta x \leq \tfrac{1}{2}n, \ \forall\, \delta \in \{\pm 1\}^n\}.$$

For simplicity we call $\delta \in \{\pm 1\}^n$ a facet. Also, consider the half line $r = \{\bar{x} + \lambda a : \lambda \geq 0\}$ originating at a point $\bar{x} \in \text{int}\, K^*$, with direction $a \in I\!R^n$. First, note that r intersects a facet δ for the value $\Lambda(\delta) > 0$ if and only if

$$(\bar{x} + \Lambda(\delta)a)\delta \quad = \quad \tfrac{1}{2}n$$

and

$$\Lambda(\delta) \quad = \quad \frac{n/2 - \delta\bar{x}}{\delta a} \ > 0.$$

At this point we must introduce some notation. For $\delta \in \{\pm 1\}^n$ and $I \subseteq N$ we

define

$$\begin{array}{lcllcl} p(\delta, I) & := & -\sum_{i \in I} \delta_i \bar{x}_i & , & P(\delta) & := & n/2 + p(\delta, N), \\ q(\delta, I) & := & \sum_{i \in I} \delta_i a_i & , & Q(\delta) & := & q(\delta, N). \\ \lambda(\delta, I) & := & p(\delta, I)/q(\delta, I) & . & & & \end{array}$$

We also denote

$$v(i) \quad = \quad -\frac{\bar{x}_i}{a_i} \ (i = 1, \ldots, n).$$

Notice that $\bar{x} \in \text{int } K^*$ implies $P(\delta) > 0$ for all δ, and

$$\Lambda(\delta) \quad = \quad \frac{P(\delta)}{Q(\delta)}.$$

Definition 3.1 *Let δ be a facet of K^*.*

1. *We call δ* reachable, *if $Q(\delta) > 0$.*
2. *We call δ* first reachable *if it is reachable, and $\Lambda(\delta)$ is minimal.*
3. *Let u be any vector in $\mathbb{R}^n$.* Flipping *the i^{th} component of u is replacing u_i by $-u_i$. For $I \subseteq N$ we denote by $u \diamond I$ the vector obtained from u by flipping all components in I. Also, we write $u \diamond i$ for $u \diamond \{i\}$.*
4. *Let δ be a reachable facet of K^*, and $I \subseteq N$. We say that*
   (a) *I is a* feasible flip *if $\delta \diamond I$ is reachable.*
   (b) *If, in addition, $\Lambda(\delta \diamond I) < \Lambda(\delta)$, we say that I is a* decreasing flip. Increasing, nonincreasing, nondecreasing *and* constant *flips are defined in the obvious way.*
   (c) *I is a* single flip *(resp.* double flip*), if $|I| = 1$ (resp. $|I| = 2$)). Otherwise I is a* multiple flip. □

Clearly, if δ is not reachable, then either it is parallel to r (iff $Q(\delta) = 0$), or it is intersected by the ray with origin $\bar{x}$ and direction $-a$ (iff $Q(\delta) < 0$). If δ is a reachable facet, then

$$\Lambda(\delta \diamond I) \quad = \quad \frac{P(\delta) - 2p(\delta, I)}{Q(\delta) - 2q(\delta, I)}.$$

The following technical lemma will be used throughout this section; it can be proved by simple cross-multiplication.

Lemma 3.2 *Let p_1, q_1, p_2, q_2 be real numbers, $q_1 > 0$, $q_2 > 0$. Then the following hold.*

(Average rule) $\frac{p_1}{q_1} < \frac{p_2}{q_2} \iff \frac{p_1+p_2}{q_1+q_2} < \frac{p_2}{q_2}$.

(Left subtraction rule) *If* $q_1 - q_2 > 0$, *then* $\frac{p_1}{q_1} < \frac{p_2}{q_2} \iff \frac{p_1-p_2}{q_1-q_2} < \frac{p_1}{q_1}$.

(Right subtraction rule) *If* $q_2 - q_1 > 0$, *then* $\frac{p_1}{q_1} < \frac{p_2}{q_2} \iff \frac{p_2}{q_2} < \frac{p_2-p_1}{q_2-q_1}$. □

The following theorem gives a necessary and sufficient condition for the existence of decreasing flips.

Theorem 3.3 *Let* δ *be a reachable facet of* K^*, *and* $I \subseteq N$. *Then* I *is a decreasing flip if and only if one of the following conditions hold:*

- $q(\delta, I) > 0$ *and* $\Lambda(\delta) < \lambda(\delta, I)$;
- $q(\delta, I) < 0$ *and* $\Lambda(\delta) > \lambda(\delta, I)$;
- $q(\delta, I) = 0$ *and* $0 < p(\delta, I)$.

Proof As δ is fixed, for brevity we omit it from the above symbols; i.e. we denote $P = P(\delta),\ \Lambda = \Lambda(\delta),\ q(I) = q(\delta, I),$ etc. First, assume $q(I) > 0$. We shall prove that I is a decreasing flip if and only if $\Lambda < \lambda(I)$.

(If): The inequality $\Lambda < \lambda(I)$ is equivalent to

$$\frac{P}{Q} \quad < \quad \frac{2p(I)}{2q(I)}. \tag{3.2}$$

As $\bar{x} \in \text{int } K^*,\ P(\delta \diamond I) = P - 2p(I) > 0$, hence

$$P \quad > \quad 2p(I). \tag{3.3}$$

By (3.2) and (3.3) we get $Q > 2q(I)$, that is

$$Q - 2q(I) > 0. \tag{3.4}$$

Hence I is a feasible flip, and applying the left subtraction rule to (3.2) yields

$$\frac{P - 2p(I)}{Q - 2q(I)} \quad < \quad \frac{P}{Q}\ , \tag{3.5}$$

that is I is a decreasing flip.

(Only if) : Suppose that I is a decreasing flip, equivalently (3.4) and (3.5) above hold. By the right subtraction rule we get

$$\Lambda \ = \ \frac{P}{Q} \quad < \quad \frac{2p(I)}{2q(I)} \ = \ \lambda(I) \tag{3.6}$$

Second, assume $q(I) < 0$. Then $\lambda(I) < \Lambda$ if and only if

$$\frac{-p(I)}{-q(I)} < \frac{P}{Q}. \tag{3.7}$$

Also I is a decreasing flip if and only if

$$\frac{P - 2p(I)}{Q - 2q(I)} < \frac{P}{Q} \tag{3.8}$$

By the average rule, (3.7) and (3.8) are equivalent.

The case $q(I) = 0$ is straightforward to check. □

To make our presentation more compact, we introduce the following notation. We define the ratio $p/0$ by

$$\frac{p}{0} = \begin{cases} +\infty \text{ if } p > 0 \\ -\infty \text{ if } p \leq 0 \end{cases}$$

Then Theorem 3.3 can be rewritten as

Theorem 3.3' *Let δ be a reachable facet of K^*, and $I \subseteq N$. Then I is a decreasing flip if and only if one of the following conditions holds:*

- $q(\delta, I) \geq 0$ *and* $\Lambda(\delta) < \lambda(\delta, I)$ *;*
- $q(\delta, I) < 0$ *and* $\Lambda(\delta) > \lambda(\delta, I)$. □

Remark 3.4 Notice that in Theorem 3.3' it suffices to check the ratio $\lambda(\delta, I)$ to decide whether I is a feasible *and* decreasing flip. Simple cross-multiplication proves that I is a constant flip, iff $p(\delta, I) = q(\delta, I) = 0$ or $\Lambda(\delta) = \lambda(\delta, I)$ regardless of $q(\delta, I)$'s sign. Therefore, when, for an I with $q(\delta, I) \neq 0$, the opposite inequality between $\Lambda(\delta)$ and $\lambda(\delta, I)$ holds in Theorem 3.3' then I is either an infeasible or an increasing flip. □

Theorem 3.5 *Let δ be a reachable facet of K^*, I and J disjoint subsets of N. Assume that $I \cup J$ is a decreasing flip for δ. Then either I or J is a decreasing flip for δ.*

Proof We distinguish three cases (the case when either one of $q(\delta, I)$ or $q(\delta, J)$ is zero is straightforward to check), and use a short notation, as in the previous theorem.

Case 1 $q(I) > 0$ and $q(J) > 0$. Suppose that neither I nor J is decreasing, i.e.

$$\Lambda \geq \frac{p(I)}{q(I)} \text{ and } \Lambda \geq \frac{p(J)}{q(J)}. \tag{3.9}$$

By the average rule then

$$\Lambda \geq \frac{p(I \cup J)}{q(I \cup J)}, \tag{3.10}$$

a contradiction.

Case 2 $q(I) < 0$ and $q(J) < 0$. The proof is similar to Case 1.

Case 3 $q(I) > 0$ and $q(J) < 0$. Suppose that neither I nor J is decreasing, i.e.

$$\frac{p(I)}{q(I)} \leq \frac{P}{Q} = \Lambda \leq \frac{-p(J)}{-q(J)}. \tag{3.11}$$

Then by the average rule (with $p_1/q_1 = P/Q,\ p_2/q_2 = -p(J)/-q(J)$)

$$\frac{p(I)}{q(I)} \leq \frac{P}{Q} \leq \frac{P - 2p(J)}{Q - 2q(J)}, \tag{3.12}$$

and by the right subtraction rule (with $p_1/q_1 = p(I)/q(I),\ p_2/q_2 = (P - 2p(J))/(Q - 2q(J))$) we obtain

$$\frac{P}{Q} \leq \frac{P - 2p(I \cup J)}{Q - 2q(I \cup J)}, \tag{3.13}$$

contradicting the assumption that $I \cup J$ is a decreasing flip.

Case 4 $q(I) < 0$ and $q(J) > 0$. Follows by symmetry. □

Example 2.1 continued For δ^3 $I = \{1, 2\}$ is a decreasing double flip. By Theorem 3.5 at least one of $\{1\}$ and $\{2\}$ must be decreasing. Indeed, $\{2\}$ is a decreasing flip, but $\{1\}$ is increasing. □

The following assumptions shall remain in force throughout the rest of the paper.

$$a \geq 0,\ v(1) \geq ... \geq v(n) \text{ and no index } i \text{ with } p(i) = q(i) = 0 \text{ exists.} \tag{3.14}$$

We do not lose generality by assuming $a \geq 0$ as flipping any component of *both* $\bar{x}$ and a yields a problem isomorphic to the original one. Precisely, δ is intersected by r for the value Λ if and only if $\delta \diamond i$ is intersected for the same value by the ray with origin $\bar{x} \diamond i$ and direction $a \diamond i$. Assuming that there is no index i with $p(i) = q(i) = 0$ is also non restrictive, since such a component may be removed from the problem, the value chosen for δ_i being irrelevant.

Using these assumptions and specializing Theorem 3.3' we obtain

Corollary 3.6 *Let δ be a reachable facet of K^*. Then*

(1) $i \in N$ is a decreasing flip if and only if one of the following conditions hold:

(1.1) $\delta_i = +1$ and $v(i) > \Lambda(\delta)$;

(1.2) $\delta_i = -1$ and $v(i) < \Lambda(\delta)$.

(2) δ is first reachable if and only if there is no decreasing single flip for δ.

Proof (1) follows by the characterization of decreasing flips in Theorem 3.3'. In (2) the "only if" part is obvious. To prove "if" assume that δ is not first reachable and it differs from a first reachable facet in the index set $I = \{i_1, \ldots, i_h\}$. As I is a decreasing flip in δ, by Theorem 3.5 at least one of the i_j's must be a decreasing flip. □

In the following, we state two algorithms: one for finding the intersection of r with the boundary of K^*, the second one for enumerating the first k extended facets of K^* intersected by r. These are improvements upon the corresponding algorithms proposed in [3].

To find a first reachable facet, we consider a simple algorithm that, given a reachable facet δ repeatedly applies a series of decreasing flips:

Algorithm First-Facet
Let δ be a reachable facet of K^*.
while (there is a decreasing single flip i for δ)
set $\delta = \delta \diamond i$.
end while
end

Lemma 3.7 *The following hold.*

(1) For an arbitrary initial δ Algorithm First-Facet terminates in at most $2n$ iterations.

(2) If the initial δ is set to e, then it terminates in at most n iterations, and can be implemented to run in $O(n)$.

Proof A component of the current δ is flipped if it is in one of states (1.1) or (1.2) of Corollary 3.6. During the algorithm $\Lambda(\delta)$ keeps decreasing. So, if δ_i is flipped in state (1.1) then it will not be flipped again. If δ_i is flipped in state (1.2) then we may flip it again in state (1.1), when $\Lambda(\delta)$ becomes less than $v(i)$. Hence every component of δ changes at most twice. This argument proves (1). If $\delta = e$, and the consecutive flips are $1, 2, \ldots$ then all components are flipped at most once. Also, we can find the next decreasing flip in $O(1)$ time per iteration. This implies (2). □

Example 2.1 continued Starting at δ^4 if we first flip component 1 from $-$ to $+$ then en route to δ^1 it will be flipped back to $-$. However, if we flip component 2 first, then we immediately get the first reachable δ^1. □

Now we turn to the problem of enumerating the first k facets of K^* intersected by r. Let δ^* be a fixed first reachable facet. Define the auxiliary weighted digraph $G = (V, A, w)$ as

$$\begin{aligned} V &= \{\, \delta \in \{\pm 1\}^n \,:\, Q(\delta) > 0 \,\}, \\ A &= \{\, (\delta, \delta \diamond i) \,:\, \Lambda(\delta) \leq \Lambda(\delta \diamond i) \,\}, \\ w(\delta, \delta \diamond i) &= \Lambda(\delta \diamond i) - \Lambda(\delta) \text{ for } (\delta, \delta \diamond i) \in A. \end{aligned}$$

Note that G is weakly connected : if $Q(\delta^1)$ and $Q(\delta^2)$ are positive, then we can arrange the flips leading from δ^1 to δ^2 so that $Q(\delta)$ is also positive for every intermediate δ. Moreover, by Corollary 3.6 for all $\delta \in V \setminus \{\delta^*\}$ there exists a directed path from δ^* to δ. (This path can be chosen as the reverse of the path obtained by first performing decreasing flips starting at δ and leading to an arbitrary first reachable facet, then performing constant flips leading to δ^*.) Therefore, finding the first k facets of the octahedron intersected by r is equivalent to finding the k nodes of G whose distance from δ^* is minimal. Although G is of exponential size, we can find δ^* in time linear in n, and given any δ we can generate each of its neighbors in G in constant time.

The following algorithm, which is merely a restatement of Dijkstra's algorithm for finding a shortest path tree in a digraph with nonnegative arc-weights, (see for example [21]), finds the k nodes of G whose distance from δ^* is minimal.

Algorithm Enumerate
Label δ^* with $\Lambda(\delta^*)$. Set **count**=0 .
while (**count** $<$ k)
    Select a labeled, unscanned node δ, with $\Lambda(\delta)$ minimal.
    Scan δ, by generating each δ' such that $(\delta, \delta') \in A$,
    and assigning it the label $\Lambda(\delta')$, if it has not been labeled before.
    Set **count = count + 1**
end while

Since for every $\delta \in V$ the length of *every* path from δ^* to δ is $\Lambda(\delta) - \Lambda(\delta^*)$, there is no need to update labels. We now study the computational complexity of Algorithm 2.

Theorem 3.8 *If $\delta \neq \delta'$ implies $\Lambda(\delta) \neq \Lambda(\delta')$, then Algorithm Enumerate can be implemented in $O(kn \log k)$ time.*

Proof We keep the labeled nodes in a balanced binary tree T. In T, a node δ is identified by $\Lambda(\delta)$ and its description consists of a pointer to its parent among the scanned nodes, and the index in which they differ. Since we need only the k nodes

closest to δ^*, the size of T can be kept at most k. Thus the cost of an individual operation on T (such as checking membership, insertion or deletion) is $O(\log k)$.

Since every node in G has $O(n)$ neighbors, the total time taken by the operations performed on T is $O(n \log k)$ in one iteration of the **while** loop, and $O(kn \log k)$ in the entire algorithm. We generate the full description of only k nodes which are scanned during the algorithm, at the cost of $O(kn)$. The claimed complexity follows. □

The assumption of Theorem 3.8 fails to hold whenever the ray r intersects more than one facet of K^* in the same point. This case is taken care of by the algorithm of the next section.

3.2 A reverse search algorithm

In this subsection we describe a more efficient algorithm for the facet enumeration problem based on the *reverse search* paradigm of Avis and Fukuda [1]. Informally, inside the **while** loop of Algorithm Enumerate we shall only generate nodes δ' which have not been previously labeled; in other words, we never examine arcs connecting two nodes of the partially built shortest path tree. This approach has the following advantages:

- If there are different facets at the same distance from δ^*, i.e. the assumption of Theorem 3.8 fails, in Algorithm Enumerate we can no longer identify the nodes by this distance. In this case (since comparing two n-bit numbers is an operation of cost $O(n)$), the complexity of the simple enumeration algorithm becomes $O(kn^2 \log k)$. Using reverse search we can retain the $O(kn \log k)$ complexity.
- Even if there are no facets at the same distance from δ^*, the reverse search algorithm is considerably faster in practice.

Before formally describing the reverse search algorithm we give two examples to illustrate each one of these issues. For a randomly chosen ray the probability of hitting two extended facets for the same Λ value is zero. However, for certain rays, intersecting a large number of extended facets at the same distance is the rule rather than the exception, as shown by the following example.

Example 3.9 Consider the LP-relaxation of *(IP)*

$$P \quad = \quad \{x \mid Ax \geq b \;\; 0 \leq x \leq e\}$$

where A is $m \times n$. Let x be a vertex of P, and a a direction pointing from x to an adjacent vertex. Let $\bar{x} = x - \frac{1}{2}e$, and δ^* a fixed facet of K^* intersected first by r. Then,

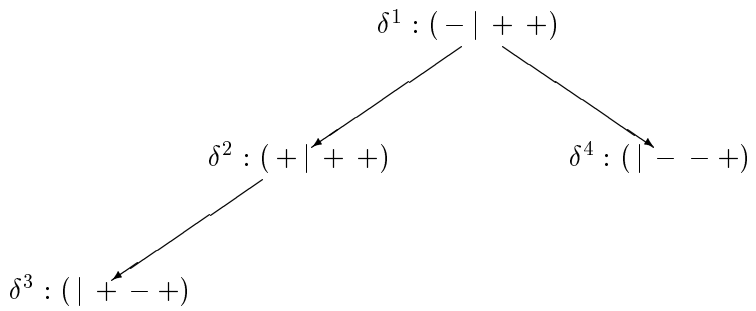

Figure 2: A shortest path tree of Example 1.1

after a possible permutation of components, $\bar{x}$, a and δ^* will have the form

$$\begin{array}{rcccccccccc} & & & & \overset{1.}{} & & & & \overset{\ell.}{} & & \\ \bar{x} & = & (& -1/2 & \dots & -1/2 & 1/2 & \dots & 1/2 & \dots &) \\ a & = & (& 0 & \dots & 0 & 0 & \dots & 0 & \dots &) \\ \delta^* & = & (& - & \dots & - & + & \dots & + & \dots &) \end{array}$$

where $\ell \geq n - m$. This can be seen as follows: as x is a vertex, at least $n - m$ of its components are at their lower or upper bounds. Also, a must have zeros in all the corresponding components, except for one.

Let $t \geq 1$. If we flip *any* t components of δ^* among the first ℓ, the resulting facets will all have the same Λ value. That is, there will be groups of extended facets with cardinality

$$\binom{\ell}{1}, \binom{\ell}{2}, \dots \binom{\ell}{l}$$

at the same distance from δ^*. If $n - m$ (hence also ℓ) is large, then checking the corresponding $0 - 1$ points for feasibility is computationally prohibitive even for the facets of the second group. As the rays are exactly of this form when generating enumerative intersection cuts, we must conclude, that these cuts cannot be made very deep.

Example 3.10 The shortest path tree produced by Algorithm Enumerate for all reachable facets in Example 2.1 is shown in Figure 2 (for the moment the reader may ignore the vertical bars; they will only be used later). Note that even in this small example there is an arc in the digraph G, namely (δ^3, δ^4) that is not an arc of the shortest path tree. The aim of the Reverse Search Procedure, to be described next, is precisely to disregard such arcs.

Let δ^* be a fixed first reachable facet of K^*. We introduce a function

$$f : V \setminus \{\delta^*\} \longrightarrow V,$$

to be defined below, such that the graph $G_f = (V, A_f)$ where $A_f = \{(\delta, \delta \diamond i) : f(\delta \diamond i) = \delta\}$, is a subgraph of G and an arborescence rooted at δ^*. We then show that f can be efficiently *reversed*, i.e. in Algorithm Enumerate one can generate only those nodes $\delta \diamond i \in V$ that satisfy $(\delta, \delta \diamond i) \in A_f$, using only local information at δ.

We introduce the following notation. We denote by $\mathrm{dec}_+(\delta)$ the set of decreasing $+$ to $-$ flips in δ. Similarly, $\mathrm{noninc}_-(\delta)$ denotes the set of nonincreasing $-$ to $+$ flips in δ.

The definition of f is

if $\mathrm{dec}_+(\delta) \neq \emptyset$
  Let $i = \min\{j \mid j \in \mathrm{dec}_+(\delta)\}$.
  Set $f(\delta) = \delta \diamond i$.
elseif $\mathrm{noninc}_-(\delta) \neq \emptyset$
  Let $i = \max\{j \mid j \in \mathrm{noninc}_-(\delta)\}$.
  Set $f(\delta) = \delta \diamond i$.
else $f(\delta)$ is undefined.
end if

Theorem 3.11 *The following hold:*

(1) There is a unique first reachable facet δ^*, for which f is undefined.

(2) The graph G_f is an arborescence rooted at δ^*.

Proof Denote by Λ^* the value of λ for which r intersects the first facet of K^*. By Corollary 3.6 and assumption (3.14), a facet δ is first reachable, if and only if it is of the form

$$\delta \quad = \quad (- \ldots \overset{s}{-} \pm \ldots \overset{t}{\pm} + \ldots +),$$

where

$$v(i) \qquad \begin{cases} > \Lambda^* & \text{if } i \leq s \\ = \Lambda^* & \text{if } s+1 \leq i \leq t \\ < \Lambda^* & \text{if } i \geq t+1. \end{cases}$$

If δ is among these facets, then $f(\delta)$ is undefined if and only if all the ambiguous components (i.e. those market $\pm$) are $+$, hence part (1) of the claim follows. Also, $\Lambda(f(\delta)) = \Lambda(\delta)$ if and only if δ is first reachable. In this case, f flips the largest index between $s+1$ and t, hence G_f cannot contain a circuit, i.e. G_f is an arborescence, as required. □

Example 2.1 continued The shortest path tree of Figure 2 is the same as G_f. At a facet δ on the left side of the vertical bar for all indices i $v(i) > \Lambda(\delta)$ and the opposite inequality holds for all indices on its right side. In δ^4 the decreasing flips are 1 and 2. Flipping 2 we obtain δ^1 the parent of δ^4 in the tree. □

Finally we show how to *reverse* f, that is, given a reachable facet δ of K^*, how to compute all indices i that satisfy $f(\delta \diamond i) = \delta$ in $O(n)$ time. Clearly, $f(\delta \diamond i) = \delta$ if and only if one of the conditions (1) and (2) below hold:

(1) (1.1) $i \in \mathrm{dec}_+(\delta \diamond i)$
(1.2) $\{1, \ldots, i-1\} \cap \mathrm{dec}_+(\delta \diamond i) = \emptyset$.

(2) (2.1) $i \in \mathrm{noninc}_-(\delta \diamond i)$
(2.2) $\{i+1, \ldots, n\} \cap \mathrm{noninc}_-(\delta \diamond i) = \emptyset$.
(2.3) $\mathrm{dec}_+(\delta \diamond i) = \emptyset$

Now, assume that min-plus(δ) is a field added to the description of δ in Algorithm Enumerate that contains the smallest index of a + component of δ. Then we consider the following

Algorithm Reverse-f
Input : A reachable facet δ of K^*.
Output : Set of indices i satisfying $f(\delta \diamond i) = \delta$.
Set $i = 1$.
while ($\delta_i = -1$ and $v(i) > \Lambda(\delta)$)
    Output i.
    Set $i = i + 1$.
end while
Set $i = n$.
while ($\delta_i = +1$ and $v(i) \leq \Lambda(\delta)$)
    if ($Q(\delta \diamond i) > 0$ and $v(\text{min-plus}(\delta)) \leq \Lambda(\delta \diamond i)$)
        Output i.
    end if
    Set $i = i - 1$.
end while

Lemma 3.12 *Algorithm* **Reverse-f** *is correct.*

Proof First we show that the indices output in the first **while** loop are exactly the ones that satisfy (1). Clearly, (1.1) holds iff i is an increasing $-$ to $+$ flip in δ , that is

$$\delta_i = -1 \text{ and } v(i) > \Lambda(\delta).$$

By the nonascending order of the $v(i)$'s, (1.2) is equivalent to

$$\delta_1 = \ldots \delta_{i-1} = -1,$$

thus our claim follows.

Second, we show that the indices output in the second **while** loop are exactly the ones that satisfy (2). Clearly, (2.1) is true iff i is a nonincreasing + to − flip in δ, that is

$$\delta_i = +1,\ v(i) \leq \Lambda(\delta) \text{ and } Q(\delta \diamond i) > 0.$$

Again, by the nonascending order of the $v(i)$'s (2.2) is equivalent to

$$\delta_{i+1} = \ldots \delta_n = +1,$$

and (2.3) is equivalent to

$$v(\text{min-plus}(\delta \diamond i)) \quad \leq \quad \Lambda(\delta \diamond i). \tag{3.15}$$

If $i = \text{min-plus}(\delta)$ then $\text{min-plus}(\delta \diamond i) = i + 1$ and (3.15) is trivially true, as

$$\lambda(\delta, i+1) \leq \lambda(\delta, i) \leq \Lambda(\delta) \leq \Lambda(\delta \diamond i).$$

If $i \neq \text{min-plus}(\delta)$ then $\text{min-plus}(\delta \diamond i) = \text{min-plus}(\delta)$. The correctness of our algorithm follows. □

Corollary 3.13 *Suppose that in the* **while** *loop of Algorithm Enumerate we only generate facets $\delta' = \delta \diamond i$ that satisfy $f(\delta') = \delta$. Then the algorithm can be implemented in $O(kn \log k)$ time.*

Proof Straightforward; we must only notice that $\text{min-plus}(\delta \diamond i)$ can be computed in constant time from $\text{min-plus}(\delta)$ when $\delta \diamond i$ is added to the list of labeled nodes. □

Finally, we outline a modification of the reverse search algorithm to obtain an additional speedup. Let δ^* be a fixed first reachable facet of K^* (found by Algorithm First-Facet). In the definition of the function f it is then sufficient to consider only those flips in which δ and δ^* differ. That is, we define f^* by simply replacing $\text{dec}_+(\delta)$ and $\text{noninc}_-(\delta)$ in the definition of f by their intersection with $\{i \mid \delta_i \neq \delta_i^*\}$. By Corollary 3.6 the only reachable facet for which f^* is undefined is δ^*. The only components of a reachable facet δ that are flipped by f^* are the ones different from δ^*, hence on any path from δ to δ^* the hamming-distance from δ^* keeps decreasing. Therefore Theorem 3.11 holds with f replaced by f^*. Also, we can compute the indices i s.t. $f^*(\delta \diamond i) = \delta$ by an algorithm similar to Algorithm Reverse-f by restricting the indices to consider to the set $\{i | \delta_i = \delta_i^*\}$.

4 The expanded octahedron

One of the main drawbacks of the above described algorithm is that every $0-1$ point enumerated must be checked for feasibility, a time consuming operation when the linear system of (IP) is large. Moreover, our procedure does not use the knowledge of the constraint set. We could envision using that information in the choice of the direction a, as we will do in Section 5, but it would also be desirable to avoid, by other means, the enumeration of facets of the octahedron that lead to infeasible $0-1$ points. We now show that this is indeed possible, without a significant increase in the complexity of the results explained so far. For this we use a construction introduced by Balas and Zoltners in [8].

Definition 4.1 *The inequality*

$$z_0 \leq sx \leq s_0 \tag{4.16}$$

is a canonical inequality *(CI for short) if* $s \in \{0, \pm 1\}^n$ *and* s_0 *and* z_0 *are integers, with* $z_0 \leq s_0$*, (one of* $z_0 = -\infty$*,* $s_0 = +\infty$ *is allowed). If* $z_0 = s_0$*, then (4.16) is called a* canonical equality *(CE for short).* □

We are interested only in *valid* canonical inequalities (equations), i.e. canonical inequalities satisfied by all feasible 0-1 points.

Let

$$z_0^i \leq s^i x \leq s_0^i \quad (i = 1, \ldots, t) \tag{4.17}$$

be a system of canonical inequalities, where the s^i 's have disjoint support.

Lemma 4.2 *The polytope*

$$Q = \{0 \leq x \leq e \mid x \text{ satisfies } (4.17)\}$$

has only $0-1$ *vertices.*

Proof The coefficient matrix of the system (4.17) is totally unimodular. □

We call Q a *truncated cube.* It is the convex hull of those vertices of K satisfying (4.17).

We shall be interested in facets of K^* whose corresponding $0-1$ points satisfy (4.17). The correspondence between $0-1$ points x and facets δ of K^* is given by $\delta = 2x - e$. Thus the vertex x of K satisfies the system (4.17) if and only if the corresponding facet δ of K^* satisfies

$$l_0^i \leq s^i \delta \leq u_0^i \quad (i = 1, \ldots, t) \tag{4.18}$$

with $l_0^i = 2z_0^i - s^i e,\ u_0^i = 2s_0^i - s^i e$.

We call the polytope

$$Q^* \quad = \quad \{x \in I\!R^n \ \mid \delta x \leq \frac{1}{2}n \ \text{ for all } \ \delta \in \{\pm 1\}^n \ \text{ satisfying } \ (4.18)\}$$

an *expanded octahedron*. Q^* is the outer polar of Q, as defined in [4]. It can be obtained from K^* by the removal of those facets containing vertices of K violating (4.17).

We define a facet of Q^* to be reachable, first reachable and a flip to be feasible, decreasing, increasing, etc. by simply replacing K^* by Q^* in Definition 3.1. We consider the problem of finding a first reachable facet of Q^* and enumerating the first k facets intersected by r.

To simplify the exposition we restrict ourselves to the case when there is only *one* canonical inequality. Generalizing the results for the case of more than one inequality is straightforward, at the cost of introducing more cumbersome notation. We denote the inequality (4.18) in the generic fashion

$$l \leq \quad s\delta \quad \leq u. \tag{4.19}$$

We fix δ^*, a first reachable facet of Q^*.

Theorem 4.3 *Let δ be a reachable facet of Q^*, and assume*

$$\begin{aligned} \{p_1, \ldots, p_v\} &= \{i \mid \delta_i \neq \delta_i^*,\ \delta_i s_i = +1\}, \\ \{m_1, \ldots, m_w\} &= \{i \mid \delta_i \neq \delta_i^*,\ \delta_i s_i = -1\}. \end{aligned}$$

Then one of the assertions (1) and (2) is true.

(1) δ is first reachable.

(2)(2.1) There is a decreasing single flip in δ;

(2.2) There is a decreasing double flip in δ of the form (p_l, m_l) for some $l \leq \min\{v, w\}$.

Proof As (1) and (2) cannot hold simultaneously, we assume that (1) is false, and prove that either (2.1) or (2.2) must hold. Denote

$$N_{\neq}(\delta) \quad = \quad \{i \mid \delta_i \neq \delta_i^*\}.$$

We distinguish two cases.

Case 1 The inequality (4.19) is not tight for δ. As $N_{\neq}(\delta)$ is a decreasing flip for δ, Theorem 3.5 shows that there exists $i \in N_{\neq}(\delta)$ with $\Lambda(\delta \diamond i) < \Lambda(\delta)$. Also, by the

argument in Lemma 4.2, and the one-to-one correspondence of vertices of K and facets of K^* , $\delta \diamond i$ cannot violate the inequality (4.19), hence i is a decreasing flip.

Case 2 The inequality (4.19) is tight for δ. Assume $l < s\delta = u$ (an analogous argument holds when $l = s\delta < u$, or $l = s\delta = u$). In this case a single flip may result in violating the inequality (4.19). As flipping p_i for $i = 1, \ldots, v$ decreases $s\delta$ and flipping m_j for $j = 1, \ldots, w$ increases it, we must have $w \leq v$. Partition $N_{\neq}(\delta)$ as

$$\{p_1, m_1\} \cup \ldots \cup \{p_w, m_w\} \cup \{p_{w+1}\} \cup \ldots \{p_v\} \cup \{z_1\} \cup \ldots \cup \{z_j\}$$

where

$$\{z_1, \ldots, z_j\} \quad = \quad \{i \in N_{\neq}(\delta) \mid \ s_i = 0\}$$

We can perform *any* of the above single or double flips without violating (4.19). By Theorem 3.5 at least one of these flips decreases $\Lambda(\delta)$, thus the required claim follows. □

Using Theorem 4.3, it is straightforward to extend Algorithm First-Facet to find the first facet of Q^* intersected by a half line: we must start with a reachable facet of Q^* and perform a decreasing single or double flip as long as one exists. Contrary to the case of the octahedron, there is no obvious bound on the running time of this procedure. Balas and Zoltners [8] describe a different algorithm (not based on single or double flips) with worst-case complexity $O(n^2)$. In our experience our method works well in practice.

Modifying Algorithm Enumerate is also straightforward. As in the previous section, we define the auxiliary weighted digraph $G = (V, A, w)$ as

$$\begin{aligned} V &= \{\, \delta \in \{\pm 1\}^n : \ l \leq s\delta \leq u, \ Q(\delta) > 0 \,\}, \\ A &= \{\, (\delta, \delta \diamond I) : \ 1 \leq |I| \leq 2 \ \text{ and } \Lambda(\delta) \leq \Lambda(\delta \diamond I) \,\}, \\ w(\delta, \delta \diamond I) &= \Lambda(\delta \diamond I) - \Lambda(\delta) \text{ for } (\delta, \delta \diamond I) \in A. \end{aligned}$$

The number of nondecreasing double flips in a reachable facet of Q^* is $O(n^2)$. Thus the running time of the algorithm is $O(kn^2 \log n)$, if there are no two facets with the same Λ value, and $O(kn^3 \log n)$ otherwise.

Finally, we outline a reverse search algorithm to enumerate the facets of Q^*. We define a function

$$g : V \setminus \{\delta^*\} \longrightarrow V$$

such that the graph $G_g = (V, A_g)$ where $A_g = \{(\delta, \delta') : g(\delta') = \delta\}$ is a subgraph of G and an arborescence rooted at δ^*. We also describe a procedure to reverse g.

First we give an informal description of g. If there is a single decreasing, or nonincreasing flip for δ, we choose one as in the case of the octahedron. If there is no such

flip, then we consider the restricted set of double flips described in Theorem 4.3. If there is a decreasing double flip among them with a nonnegative value of the resulting q, then we choose the one with the maximal λ. If not, then we choose one with a negative q value having a minimal λ. We break ties using lexicography. That is, the preference among the restricted set of double flips is given by the same rule, as the preference among the single flips.

For the formal definition of g we need some more notation. For δ, a reachable facet of Q^*, we denote

$$\begin{aligned}
\mathrm{dec}_+(\delta) &= \{i \mid \delta_i \neq \delta^*_i, \text{ and } i \text{ is a decreasing } + \text{ to } - \text{ flip for } \delta\},\\
\mathrm{noninc}_-(\delta) &= \{i \mid \delta_i \neq \delta^*_i, \text{ and } i \text{ is a nonincreasing } - \text{ to } + \text{ flip for } \delta\},\\
\mathrm{dec}_{+,2}(\delta) &= \{I \mid I \text{ is a decreasing double flip of the form } (p_\ell, m_\ell), \text{with } q(\delta, I) > 0\},\\
\mathrm{noninc}_{-,2}(\delta) &= \{I \mid I \text{ is a nonincreasing double flip of the form } (p_\ell, m_\ell), \text{ with } q(\delta, I) \leq 0\}.
\end{aligned}$$

Here (p_ℓ, m_ℓ) is the notation introduced in Theorem 4.3. The definition of g is given below:

if $\mathrm{dec}_+(\delta) \neq \emptyset$
  Let $i = \min\{j \mid j \in \mathrm{dec}_+(\delta))\}$.
  Set $g(\delta) = \delta \diamond i$.
elseif $\mathrm{dec}_-(\delta) \neq \emptyset$
  Let $i = \max\{j \mid j \in \mathrm{dec}_-(\delta)\}$.
  Set $g(\delta) = \delta \diamond i$.
elseif $\mathrm{dec}_{+,2}(\delta) \neq \emptyset$
  Let $I =$ the first element of $\mathrm{dec}_{+,2}(\delta)$.
  Set $g(\delta) = \delta \diamond I$.
elseif $\mathrm{dec}_{-,2}(\delta) \neq \emptyset$
  Let $I =$ the last element of $\mathrm{dec}_{-,2}(\delta)$.
  Set $g(\delta) = \delta \diamond I$.
else g is undefined.
end if

The same argument as the one used for f^* proves that G_g is an arborescence rooted at δ^*. Finally, we show how to *reverse* g, that is, given δ a reachable facet of Q^* how to compute all single or double flips I that satisfy $g(\delta \diamond I) = \delta$ in $O(n^2)$ time. The algorithm is a straightforward extension of Algorithm Reverse-f, that requires some extra bookkeeping. We have that $g(\delta \diamond I) = \delta$ if and only if exactly one of the conditions (1), (2), (3) and (4) below holds:

(1) $I = \{i\}$ and

(1.1) $i \in \mathrm{dec}_+(\delta \diamond i)$

(1.2) $\{1, \ldots, i-1\} \cap \mathrm{dec}_+(\delta \diamond i) = \emptyset$;

(2) $I = \{i\}$ and

(2.1) $i \in \mathrm{noninc}_-(\delta \diamond i)$

(2.2) $\{i+1, \ldots, n\} \cap \mathrm{noninc}_-(\delta \diamond i) = \emptyset$

(2.3) $\mathrm{dec}_+(\delta \diamond i) = \emptyset$;

(3) $|I| = 2$ and

(3.1) $I \in \mathrm{dec}_{+,2}(\delta \diamond I)$

(3.2) $\{J \mid \lambda(J, \delta \diamond I) > \lambda(I, \delta \diamond I)\} \cap \mathrm{dec}_{+,2}(\delta \diamond I) = \emptyset$

(3.3) $\mathrm{dec}_+(\delta \diamond I) = \emptyset$

(3.4) $\mathrm{noninc}_-(\delta \diamond I) = \emptyset$;

(4) $|I| = 2$ and

(4.1) $I \in \mathrm{noninc}_{-,2}(\delta \diamond I)$

(4.2) $\{J \mid \lambda(J, \delta \diamond I) < \lambda(I, \delta \diamond I)\} \cap \mathrm{noninc}_{-,2}(\delta \diamond I) = \emptyset$

(4.3) $\mathrm{dec}_{+,2}(\delta \diamond I) = \emptyset$

(4.4) $\mathrm{dec}_+(\delta \diamond I) = \emptyset$

(4.5) $\mathrm{noninc}_-(\delta \diamond I) = \emptyset$.

We give a high-level description of the algorithm to reverse g, then outline how it can be efficiently implemented.

Algorithm Reverse-g
Input : A reachable facet δ of Q^*.
Output : Set of single and double flips I satisfying $g(\delta \diamond I) = \delta$.
Set $i = 1$.
while ($v(i) > \Lambda(\delta)$)
    if ($\delta_i = \delta_i^* = -1$ and (1.2) holds)
    Output i.
    end if
    Set $i = i + 1$.
end while
Set $i = n$.
while ($v(i) \leq \Lambda(\delta)$)
    if ($\delta_i = \delta_i^* = +1$, $Q(\delta \diamond i) > 0$ and (2.2)-(2.3) hold)

Output i.
end if
Set $i = i - 1$.
end while
for (all I with $|I| = 2,\ \lambda(\delta, I) > \Lambda(\delta)$)
if ($\delta_I = \delta^*_I,\ q(\delta, I) < 0$ and (3.2)-(3.4) hold)
Output I.
end if
end for
for (all I with $|I| = 2,\ \lambda(\delta, I) \leq \Lambda(\delta)$)
if ($\delta_I = \delta^*_I,\ q(\delta, I) \geq 0,\ Q(\delta \diamond I) > 0$ and (4.2)-(4.5) hold)
Output I.
end if
end for

The conditions spelled out in the while loops are equivalent to conditions (1.1), ..., (4.1) respectively. First we outline how to check the condition (1.2) in the first **while** loop. This condition is true if and only if for all $j \in \{1, \ldots, i-1\}$ one of the assertions (i)-(iv) below holds:

(i) $\delta_j = -1$

(ii) $\delta_j = +1$ and $\delta^*_j = +1$

(iii) $\delta_j = +1,\ \delta^*_j = -1,\ s_j = -1$ and $s(\delta \diamond i) = u$

(iv) $\delta_j = +1,\ \delta^*_j = -1,\ s_j = +1$ and $s(\delta \diamond i) = l$.

It is trivial to find out, whether j satisfies (i) or (ii). If we encounter an index j for which (iii) holds, we set a flag to true, and, for the remaining indices, we shall allow only those flips i, which push the canonical inequality to its upper bound. This method takes care of detecting indices j satisfying condition (iii) (and (iv), using a different flag). Checking conditions (2.2)-(2.3) in the second **while** loop can be done similarly.

Second, we show a more efficient way of computing double flips I that satisfy $g(\delta \diamond I) = \delta$. To find such an I, assume again

$$\begin{aligned} \{p_1, \ldots, p_v\} &= \{i \mid \delta_i \neq \delta^*_i,\ \delta_i s_i = +1\} \\ \{m_1, \ldots, m_w\} &= \{i \mid \delta_i \neq \delta^*_i,\ \delta_i s_i = -1\} \end{aligned}$$

and that the p_j's and m_j's are in increasing order. Define

$$\begin{aligned} p_0 = m_0 &:= 0 \\ p_{v+1} = m_{w+1} &:= n + 1. \end{aligned}$$

If $I = \{i_1, i_2\}$ with $\delta_{i_1} s_{i_1} = -1$ and $\delta_{i_2} s_{i_2} = +1$ satisfies $g(\delta \diamond I) = \delta$ then for some index j between 0 and $\min\{v, w\}$

$$\begin{aligned} p_j &< i_1 < p_{j+1} \\ m_j &< i_2 < m_{j+1} \end{aligned}$$

must hold. Hence the restricted set of double flips in $\delta \diamond I$ can be constructed from the corresponding set in δ. Also, the method to check that I is the double flip chosen by g in $\delta \diamond I$ is entirely analogous to the method given for the single flips.

5 Implementation issues

In this section, we discuss several issues related to the implementation of OCTANE. Most of them are of a practical nature and our insights are based on extensive experimentation with a set of test problems. As we have seen, OCTANE requires three inputs, namely the starting point of the enumeration, the direction of the ray and the number k of intersections to compute. Additionally, if we decide to use an expanded octahedron, we have to generate a set of canonical inequalities that cuts off as many integer infeasible points as possible. Moreover, in the case where the chosen starting point has some integer entries, we may decide to perform the enumeration in the space corresponding to a face of the hypercube containing the starting point, by fixing some of the variables to the value they take in this point. When a relatively large number of entries in the starting point are integer this allows us to reduce significantly the space in which the enumeration is performed, yielding a considerable speedup. On the other hand, fixing too many of these variables may result in an infeasible ILP, implying that the enumeration might become a waste of time.

Early in our experimentation we found that using rays that start at the LP optimum and varying only their directions is not enough for finding good feasible solutions: the origin of the rays must also be varied. Thus we were led to develop OCTANE as a tool to be used within a branch-and-cut framework [9, 18, 20], by running it from different nodes of the enumeration tree. Depending on the problem at hand, the goal might be to solve the problem to optimality (and use the heuristic to hopefully find an optimum solution earlier) or to find a relatively good solution relatively quickly. In the former case, the heuristic plays a secondary role, since, for most problems of interest, proving that a given solution is optimal is almost as hard as finding a provably optimal solution. In the latter case, the heuristic plays a primary role and a crucial point is to decide from which nodes of the enumeration tree the heuristic should be called.

These goals are conflicting and there are several reasonable ways of comparing the performance for different settings. We focused on optimizing the total running time of the procedure, at the cost of "missing" some good solutions that could be found

earlier in the search tree. We embedded OCTANE into the branch-and-cut code MIPO described in [9].

5.1 The starting point for the enumeration

In order to start OCTANE, we need to select a point $\bar{x}$, in the interior of K^*. When running the heuristic within a branch-and-cut framework a natural choice is to let $\bar{x} = x - \frac{1}{2}e$, where x is the optimal solution found while solving the linear programming relaxation at the current node of the enumeration tree.

5.2 The directions for enumeration

The main purpose of enumerating the facets of the expanded octahedron along a particular ray $\bar{x} + \lambda a$, is to explore the 0-1 points that are in "a neighborhood" of $\bar{x}$ in the order given by the direction a. Intuitively, if a ray is directed towards the "inside" of the feasible region it seems more likely to lead to the enumeration of a feasible 0–1 solution. We experimented with several different directions. We found that the overall performance of the directions which go inside the feasible region dominates that of the directions which go "outside". The set of directions which we finally settled for is:

- *Average* ray: The vector defined as the average of the set of extreme rays (normalized) of the cone C defined by the current optimal basis of the LP.
- *Objective* ray: The inward normal to the objective function.
- *Difference* ray: The vector defined as the difference between the optimal solution of the linear relaxation at the current node and at the root node of the enumeration tree.
- *Average (weighted) slack* ray: The vector defined as the weighted average of the extreme rays of C corresponding to the non-basic slacks with positive reduced cost at the current fractional point. The weights are defined by the inverse of their reduced costs.

We also tested randomly generated rays, but practically no feasible solutions were found when using these directions.

5.3 The octahedron versus the expanded octahedron

An important question is whether we should use the octahedron or the expanded octahedron for facet enumeration. Clearly, the first procedure is simpler and faster, but

potentially leads to the enumeration of many facets that do not correspond to feasible points. On the other hand, working with the expanded octahedron is computationally more expensive, but since fewer facets are enumerated, the procedure might lead to an overall better performance. Our extensive computational experiments which are summarized in Tables A.1 and A.2 in the Appendix confirm that this is indeed the case.

5.4 Number of facets to be intersected

The number of facets of the expanded octahedron to be intersected along a search direction is determined as follows: If, for the first ten facets intersected, there exists an inequality in the problem formulation which is violated by all the enumerated 0-1 points, then we stop the enumeration along this half line. The rationale behind this choice is that if the first ten points generated are all cut off by a single inequality, it is very likely that no feasible point will be generated for this enumeration, a fact confirmed empirically.

Otherwise, i.e. if the first ten facets intersected are not all cut off by a single inequality, we enumerate at most $FMAX = 100$ facets along the half line.

5.5 Space for enumeration

Let x be the optimal solution while solving the LP-relaxation of *(IP)* at a node of the branch-and-cut tree. Define $F = \{i \in N \mid 0 < x_i < 1\}$ as the set of fractional variables. Given a direction a, we considered the following choices for the space where we carried out the enumeration:

- The full space $\mathbb{R}^N$.
- The space $\mathbb{R}^F$ of fractional variables.

First, we note that for any $S_1 \subseteq S_2 \subseteq N$, there is a one-to-one correspondence between the set of facets of the expanded octahedron in the space of the variables in S_1 and a proper subset of the set of facets of the expanded octahedron in the space of the variables in S_2. Hence, we pay for the reduction in dimension by losing facets, some of which might contain feasible points for *IP*. On the other hand, enumerating facets in the smaller space is computationally cheaper than in the larger space.

We also experimented with "intermediate" spaces, by adding to the fractional space a set of randomly selected variables in $N \setminus F$. As expected, there is a trade-off between the quality of the feasible points found and the time spent for the enumeration. Our

computational experiments indicate that the search should be carried out in the fractional space. In the Appendix we present detailed comparisons for the full space versus the fractional space of variables. The results are summarized in tables A.1 and A.2 of the Appendix.

5.6 Generation of canonical inequalities

For $s \in \{0,1\}^n$ we define the strength $str(s)$ of the canonical inequality (CI) corresponding to s by

$$l(s) = \min\{sx : x \in P\},\ u(s) = \max\{sx : x \in P\},\ str(s) = \frac{|supp(s)|}{u(s)} + \frac{|supp(s)|}{|supp(s)| - l(s)} \tag{5.20}$$

where $supp(s)$ is the support of s.

When $s \in \{0, \pm 1\}^n$ the strength of the corresponding CI is defined by first complementing all variables with negative coefficients, then using the above definition. Clearly, the stronger a CI, the smaller the number of infeasible points that are enumerated. Our computational experiments confirm this.

The generation of the CI's was implemented as follows. First, we extract a maximal set of disjoint canonical equations (CE's) directly from the problem formulation if such equations exist, by always picking the CE whose support is the largest among those using only variables not yet in any CE. Second, among the inequalities in the formulation, we consider the t ones with the largest support corresponding to variables not yet used. We compute the strength of the CI's obtained by taking a $\{0, \pm 1\}$ vector corresponding to the sign-pattern of these inequalities, setting to 0 all coefficients corresponding to a variable already in some CE. We also compute the strength of the CI obtained by simply setting to 1 all coefficients corresponding to remaining variables. Among these $t+1$ inequalities we pick the strongest one.

Since computing the strength of an inequality requires solving a pair of linear programs, t cannot be too large. In our current implementation, we set $t = 5$.

In this fashion, we generated some CE's when they were available, and one CI. Of course, it is possible to generate more than one of the latter, e.g. by repeating the second step. However, so far we have not found a good way of doing this.

5.7 The frequency of enumeration

Since the primary aim of using the heuristic in a branch-and-cut framework is to find feasible solutions as early as possible in the search tree, we use OCTANE heavily close to the root node and subsequently reduce its usage as we go deeper in the branch-and-cut tree. For the first five levels of the branch-and-cut tree, we use OCTANE at every

node. Subsequently, we use OCTANE once every eight nodes of the branch-and-cut tree.

6 Computational testing of OCTANE

6.1 The test bed

We tested OCTANE on a wide variety of pure 0-1 programs arising from applications. Many of our test problems are taken from MIPLIB, a publicly available library of real-world mixed integer programs compiled by Bixby, Boyd and Indovina [11]. The GAPxx problems are generalized assignment problems obtained from the OR-Library maintained by J.E. Beasley [10]. TSP43 is a 43-city asymmetric traveling salesman problem (see [9] for a more detailed discussion of this instance). Table 1 contains the characteristics of the problems used in our test bed. Apart from the fact that all these problems are pure 0-1 programs, they do not share any particular problem structure.

To assess the difficulty of these problems, we solved all of them to optimality with the branch-and-cut code MIPO using Lift-and-Project cuts [9]. The required times reported in this paper refer to seconds on an HP720 Apollo desktop workstation with 64 megabytes of memory. The linear programs were solved using CPLEX 3.0.

6.2 OCTANE within MIPO

We implemented OCTANE and tested the different options described in Section 5. Since we were interested in comparing the performance of different parameter settings of OCTANE and in comparing OCTANE to other heuristics, in our basic tests we made sure that the branch-and-cut algorithm did not gain any information on the feasible points found by any of the heuristics. This guaranteed that the nodes of the enumeration tree will be identical in all runs, independently of the chosen heuristic. We also called the different heuristics from the same nodes of the enumeration tree, preventing one of the heuristics to find the optimal solution of a problem just because it is the only heuristic that is run on a particular node.

In the Appendix we present the comparisons between running OCTANE on the octahedron versus the expanded octahedron, and in the full space of variables versus the space of fractional variables. Here we present our computational results with the version that was found the best - namely enumerating the facets of Q^* in the fractional space.

We say that a feasible solution x to (IP) is acceptable if it is within 10% of the optimal solution; that is, if x^* is the optimal solution to (IP) then we require $\frac{|cx^*-cx|}{|cx^*|} <$

Problem name	Constraints	0-1 Variables	Value of LP optimum	Value of IP optimum	Branch-and-Cut time	Branch-and-Cut nodes
BM23	20	27	20.57	34	6.14	512
GAP51	32	192	568.64	563	8.02	398
GAP61	40	256	768.22	761	149.26	1966
GAP84	56	384	1126.13	1117	778.77	5956
L152LAV	97	1989	4656.36	4722	4325.77	4624
LP4L	85	1086	2942.5	2967	64.32	190
LSEU	28	89	834.68	1120	12.59	468
MISC01	53	82	57.00	563.5	12.47	358
MISC03	95	159	1910.00	3360	48.34	646
MISC07	212	259	1415.00	2810	7353.01	20336
MOD008	6	319	290.93	307	197.35	2488
MOD010	146	2655	6532.08	6548	60.92	22
P0033	15	33	2520.57	3089	2.56	138
P0201	133	201	6875.00	7615	154.74	976
P0282	241	282	176867.50	258411	38.13	266
P0291	252	291	1705.13	5223.749	6.09	30
P0548	535	548	3142.88	8691	10572.45	12716
P2756	755	2756	2688.75	3124	4199.08	1490
PIPEX	25	48	773.75	788.263	15.33	734
SENTOY	25	60	-7839.27	-7772	14.27	180
TSP43	143	1177	5611	5620	209.61	520

Table 1: Problem characteristics

0.1. In all the results reported, we only list the acceptable feasible solutions. Also, these solutions are listed only if they improve (the gap) upon the previous solution found by at least 0.01%.

In Tables 2 and 3 we summarize the computational results with OCTANE using the expanded octahedron in the space of fractional variables. Table 2 summarizes the times and quality of the solutions found by OCTANE. We first show the time taken by branch-and-cut (B&C Time) and OCTANE (OCTANE Time), respectively. The column "OCTANE First" shows the percentage deviation from optimality of the first feasible solution found by OCTANE. The next column, "Time to first" shows the time used by B&C, with the embedded OCTANE, until this first solution was found. The next two columns contain analogous information for the best solution found by OCTANE ("OCTANE Best" and "Time to Best").

In Table 3 we compare, for all problems in our test bed, the number of nodes needed

Problem name	B & C Time (sec)	OCTANE Time (sec)	OCTANE First (% from opt)	Time to First (sec)	OCTANE Best (% from opt)	Time to Best (sec)
BM23	6.14	2.24	0.00	0.65	0.00	0.00
GAP51	8.02	5.13	0.005	4.36	0.001	7.62
GAP61	149.26	21.32	0.00	19.85	0.00	19.85
GAP84	778.77	41.22	0.01	4.22	0.00	173.06
L152LAV	4325.77	96.23	0.01	53.07	0.00	3631.16
LP4L	64.32	15.72	0.00	57.96	0.00	57.96
LSEU	12.39	2.39	0.03	6.03	0.00	10.12
MISC01	12.47	2.11	0.06	2.57	0.06	2.57
MISC03	48.34	3.86	0.08	2.81	0.00	14.68
MISC07	7353.01	93.29	***	***	***	***
MOD008	197.35	16.43	0.0003	0.01	0.00	1.16
MOD010	60.92	6.52	0.0007	50.41	0.0007	50.41
P0033	2.56	0.63	0.001	1.12	0.00	3.01
P0201	154.74	8.97	0.00	146.21	0.00	146.21
P0282	38.13	6.71	0.002	26.16	0.0008	37.82
P0291	6.09	2.68	0.02	2.50	0.00	5.95
P0548	10572.45	948.23	0.01	3288.71	0.01	3288.71
P2756	4199.08	429.07	0.03	1572.84	0.00	4213.06
PIPEX	15.33	2.29	0.001	1.15	0.00	6.01
SENTOY	14.27	2.81	0.001	2.20	0.00	11.76
TSP43	209.61	6.07	0.0008	4.14	0.00	52.24

Table 2: Computational results with Q^* in the fractional space - w.r.t. time

for Branch-and-cut (B&C) against the number of nodes (and their percentage of the total number) needed for B&C plus OCTANE to find the first and best solutions, respectively.

A short summary of our results is as follows. We computed how much time it takes B&C plus OCTANE to find a feasible solution within 10% of the optimum for the problems in the test bed. Then we divided this time with the time it takes B&C *without the heuristic* to solve the problem to optimality. The outcome is that OCTANE finds a 10%-optimal solution within 20% of the time for 9 instances; and within 50% of the time for 15 instances, out of the total of 23. The time taken by OCTANE as a percentage of the total time required by branch-and-cut to solve the problem to optimality is, on average, 17%.

We also experimented with OCTANE in a different mode, namely allowing MIPO

Problem name	B & C Nodes	OCTANE First (# of nodes)	OCTANE First (% of nodes)	OCTANE Best (# of nodes)	OCTANE Best (% of nodes)
BM23	512	0	0.00%	0	0.00%
GAP51	398	29	7.28%	104	26.13%
GAP61	1966	208	10.57%	208	10.57%
GAP84	5956	24	0.40%	1016	17.05%
L152LAV	4624	25	0.54%	3704	80.10%
LP4L	190	80	42.10%	80	42.10%
LSEU	468	96	20.51%	192	41.02%
MISC01	358	17	4.74%	17	4.74%
MISC03	646	6	0.92%	120	18.57%
MISC07	20336	***	***	***	***
MOD008	2488	0	0.00%	0	0.00%
MOD010	22	2	9.09%	2	9.09%
P0033	138	56	40.57%	120	86.95%
P0201	976	712	72.95%	712	72.95%
P0282	266	72	27.06%	112	42.10%
P0291	30	2	6.66%	13	43.33%
P0548	12716	1632	12.83%	1632	12.83
P2756	1490	304	20.40%	936	62.81%
PIPEX	734	8	1.08%	120	16.34%
SENTOY	180	0	0.00%	160	88.88%
TSP43	520	0	0.00%	152	29.23%

Table 3: Computational results with Q^* in the fractional space - w.r.t. number of nodes

to take advantage of the information provided by the heuristic for pruning the search tree, etc. On the same set of 21 MIPLIB problems, we found that the total running time of MIPO + OCTANE in this mode was smaller than the running time of MIPO without the heuristic in 12 cases, and larger in 9 cases. On the average, there was an overall gain in the running time due to OCTANE, but this gain was only on the order of 1%. The explanation for this lies in the fact that when a best first branching strategy is used, as is the case with MIPO, having better upper bounds (in a minimization problem) due to the use of a heuristic is of only limited help in proving the optimality of a solution. The main advantage of the heuristic is not that it radically reduces total computing time – which is not the case – but that it provides good solutions relatively early in the run.

With this in mind, we set out to run OCTANE on some difficult problems whose

solution to optimality would require very long runs, to explore its usefulness in finding good solutions in a reasonable time.

6.3 Computational results on more difficult problems

The problems we chose for this purpose are *cap6000, stein45, air04, air05* and *harp2* from MIPLIB. With the time limit set to 3 hours, these problems could not be solved to optimality with the version of MIPO used in our experiments. The characteristics of these problems are summarized in Table 4.

Problem name	Constraints	0-1 Variables	Value of LP optimum	Value of IP optimum
CAP6000	2176	6000	- 2451537.325	-2451377
HARP2	112	2993	- 74353341.502	- 73899798
AIR04	823	8904	55535.436	56137
AIR05	426	7195	25877.609	26374
STEIN45	331	45	22	30

Table 4: Characteristics of the more difficult problems

We first ran OCTANE (enumerating the facets of Q^* in the space of fractional variables) with the setting used in the previous experiments, i.e. enumerating at every node for the first five levels of the branch-and-cut tree, and at every 8 nodes afterwards. This way, OCTANE found acceptable feasible solutions only for *cap6000*, and *stein45*.

Next, we ran OCTANE at every node of the search tree. The results are summarized in Table 5, where the column headings are to be interpreted as in Table 2. The conclusion of this experiment is:

- On more difficult problems OCTANE must be run more frequently to find acceptable solutions.
- Nevertheless, with the exception of *air05* it does find such solutions early, with the total time spent on the enumeration not exceeding 17% of the total running time.

6.4 Comparison with Pivot-and-Complement

The Pivot-and-Complement heuristic developed by Balas and Martin [6] is one of the few available procedures in the literature for generating feasible solutions to 0-1 programs. A later version of this heuristic, called Pivot-and-Shift [7], can also handle

Problem name	B & C Time (sec)	OCTANE Time (sec)	OCTANE First (% from opt)	Time to First (sec)	OCTANE Best (% from opt)	Time to Best (sec)
CAP6000	10,800*	1131	0.02	5.13	0.02	975
HARP2	10,800*	1267	0.01	2606	0.01	2606
AIR04	10,800*	3723	0.02	3071	0.02	3071
AIR05	10,800*	2680	0.10	7575	0.10	7575
STEIN45	10,800*	281	0.07	12	0.03	120

*Time limit exceeded

Table 5: Computational results on the more difficult problems

mixed integer programs. We compared OCTANE with Pivot-and-Shift as implemented by C.H. Martin. We had 6 variants of Pivot-and-Shift at our disposal and we ran all of them. In our experience versions 1-3 behave similarly to each other, and so do versions 4-6. Therefore, when reporting the results in a column titled $P\&S1-3$ [$P\&S4-6$] we always report the *best* solution found by one of the versions 1-3 [4-6].

The results of the comparison are detailed in Tables 6 and 7. The entries in the column titled Gap (solution value) contain the values of the ratio $\frac{|cx^*-cx|}{|cx^*|}$ for the solutions x found for each problem, in the order in which they were found, with the corresponding solution value in parenthesis. The next three columns, one for each of the three heuristics compared (OCTANE, Pivot-and-Shift versions 1-3, and Pivot and Shift versions 4-6), contain the node of the search tree at which the given heuristic found the solution in question. A blank means that the solution in question was not found by the given heuristic (it was found by another one). For instance, consider problem LSEU. Heuristic $P\&S4-6$ finds a solution with relative gap 0.05 at node 12. OCTANE does not find a solution with exactly this gap; it reaches the performance level of 0.05 at node 96 by finding a feasible solution with gap equal to 0.03.

The reason for comparing the number of nodes, rather than the times, required to reach a specified gap is that the Pivot-and-Shift code available to us is the implementation of a stand-alone heuristic, complete with its own version of the simplex method, which does not lend itself easily to interfacing with MIPO; and the interface that we created is time-wise inefficient, considerably slower than OCTANE. In terms of nodes, Pivot-and-Shift performs clearly better than OCTANE on 4 problems (MISC01, GAP84, P0033 and TSP43). OCTANE has better performance on 8 of the problems (BM23, GAP51, L152LAV, LP4L, MISC07, MOD008, MOD010 and P0548) and the results are similar on the other problems (or very different for the two sets of variants of Pivot-and-Shift; problem P2756 is a case point). This shows that OCTANE is competitive with Pivot-and-Shift, even if the CPU time requirements, much larger for

Problem name	Gap (solution value)	OCTANE Nodes	P&S 1-3 Nodes	P&S 4-6 Nodes
BM23	0.03 (35)		0	0
	OPT (34)	0	26	26
GAP51	0.01 (557)		2	0
	0.007 (559)	1		
	0.004 (561)	11	3	11
	OPT (563)	16	72	23
GAP61	0.03 (739)			0
	0.02 (743)		0	14
	0.01 (751)	28	5	
	0.009 (754)		27	
	0.008 (755)	152		
	0.004 (758)		176	
	0.001 (760)	208		
	OPT (761)	432	648	15
GAP84	0.008 (1108)		0	
	0.006 (1110)	48		9
	0.005 (1111)			23
	0.002 (1115)		2	136
	OPT (1117)	1016	512	–
L152LAV	0.01 (4781)	25	++	++
	0.006 (4749)	28	++	++
	OPT (4722)	3704	++	++
LP4L	OPT (2967)	80	+++	+++
LSEU	0.05 (1170)			12
	0.03 (1157)	96		
	0.02 (1145)		28	
	0.01 (1136)		32	32
	OPT (1120)	192	–	–
MISC01	0.06 (595.0)	17	17	17
	0.02 (574.7)	–	216	216
MISC03	0.08 (3640)	6	2	2
	OPT (3360)	120	120	120
MISC07	0.03 (2895)	7504	+++	+++

Table 6: Comparison between OCTANE and Pivot-and-Shift 1.

Problem name	Gap (solution value)	OCTANE Nodes	P&S 1-3 Nodes	P&S 4-6 Nodes
MOD010	0.001 (6553)	2	+++	+++
MOD008	0.003 (308)		0	0
	OPT (307)	0	1	7
P0033	0.08 (3347)	56	0	0
	OPT (3089)	192	12	12
P0201	0.08 (8245)		3	
	0.06 (8055)		48	
	0.05 (7955)			11
	0.04 (7925)		128	22
	0.01 (7715)		704	704
	OPT (7615)	712		
P0282	0.08 (278004)			21
	0.07 (276088)		56	
	0.008 (260374)			72
	0.005 (259814)		72	
	0.003 (258184)	72		
	0.002 (258873)		144	
	0.001 (258723)	112		144
P0291	0.02 (5350)	2	2	2
	0.00 (5223.75)		13	13
	OPT (5223.0)	13		
P0548	0.006 (8744)	1632	+++	+++
P2756	0.08 (3376)		+++	0
	0.07 (3340)			
	0.03 (3227)	304		
	0.02 (3192)	680		
	0.01 (3163)	856		
	OPT (3124)	936		
PIPEX	0.002 (789.9)	8		
	OPT		112	112
SENTOY	0.01 (-7700)			0
	0.002 (-7758)	0	30	
	0.001 (-7761)		72	10
	OPT (-7772)	160		
TSP43	0.003 (5636)		2	0
	0.001 (5625)	0		0
	OPT (5620)	152	20	2

Table 7: Comparison between OCTANE and Pivot-and-Shift 2.

Pivot-and-Shift for the reason shown, are not taken into account.

6.5 Conclusions

Our computational experiments show that OCTANE is an efficient heuristic for 0–1 programs.

- OCTANE can be successfully embedded within a branch-and-cut procedure.
- OCTANE is an efficient and robust heuristic for pure 0-1 programs. One of its main advantages is that it works well on a variety of problems with different structures.
- Comparisons with the Pivot-and-Shift heuristics show that OCTANE is a competitive alternative to Pivot-and-Shift.
- For some problems, OCTANE and the Pivot-and-Shift heuristics complement each other. Using both heuristics at different places in the enumeration tree could be an interesting hybrid strategy.

Acknowledgement

This research was in part supported by the National Science Foundation through grants DMI-9424348 and DMS-9509581 and by the Office of Navel Research through contract N00014-89-J-1063.

References

[1] Avis, D. and Fukuda, K. "A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra." *Proceedings of the 7th ACM Symposium on Comput. Geometry,* North Conway, New Hampshire, 1991, 98-104.

[2] Balas, E. "Intersection Cuts - A New Type of Cutting Planes for Integer Programming." *Operations Research, 19,* 1971, 19-39.

[3] Balas, E. "Ranking the Facets of the Octahedron." *Discrete Mathematics, 2(1),* 1-15.

[4] Balas, E. "Integer Programming and Convex Analysis: Intersection Cuts from Outer Polars." *Mathematical Programming, 2,* 1972, 330-382.

[5] Balas, E., Bowman, J., Glover, F., and Sommer, D., "An Intersection Cut from the Dual of the Unit Hypercube." *Operations Research, 19,* 1971, 40-44.

[6] Balas, E. and Martin, C. "Pivot and Complement – A Heuristic for 0-1 Programming," *Management Science, 26(1),* 1980, 224-234.

[7] Balas, E. and Martin, C. "Pivot and Shift – A Heuristic for Mixed Integer Programming." GSIA, Carnegie Mellon University, 1986.

[8] Balas, E. and Zoltners, A. "Intersection Cuts from Outer Polars of Truncated Cubes." *Naval Research Logistics Quarterly, 22,* 1975, 977-996.

[9] Balas, E., Ceria, S. and Cornuéjols, G. "Mixed 0-1 Programming by Lift-and-Project in a Branch-and-Cut Framework," *Management Science, 42,* 1996, 1229-1246.

[10] J. E. Beasley, OR-Library: distributing test problems by electronic mail, *Journal of the Operational Research Society 41 (11)*, 1990, 1069-1072.

[11] Bixby, R.E., Boyd, E.A., and Indovina, R.R. "MIPLIB: A Test Set of Mixed Integer Programming Problems." *SIAM News, 16,* 1992.

[12] Burdet, C.A., "Enumerative Inequalities in Integer Programming." *Mathematical Programming, 2,* 1972, 32-64.

[13] Faaland, B.H. and Hillier, F.S., "Interior Path Methods for Heuristic Integer Programming Procedures." *Operations Research, 27,* 1979, 1069-1087.

[14] Glover, F. "Cut Search Methods in Integer Programming." *Mathematical Programming, 3,* 1972, 86-100.

[15] Glover, F. "A Note on Linear Programming and Integer Feasibility." *Operations Research, 16,* 1968, 1212-1216.

[16] Glover, F. "Heuristics for Integer Programming Using Surrogate Constraints." *Decision Sciences, 9,* 1977, 156-166.

[17] Glover, F. and Laguna, M. *Tabu Search,* Kluwer Academic Publishers, 1997.

[18] Hoffman, K. and Padberg, M. "Solving Airline Crew Scheduling Problems by Branch-and-Cut." *Management Science, 39,* 1993, 657-682.

[19] Ibaraki, T., Ohashi, T. and Mine, H. "A Heuristic Algorithm for Mixed Integer Programming Problems." *Mathematical Programming Study, 2,* 1974, 115-136.

[20] Padberg, M., and Rinaldi, G. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems." *SIAM Review, 33,* 1991, 60-100.

[21] Tarjan, R.E. "Data Structures and Network Algorithms." *SIAM,* 1983, Philadelphia.

Appendix

In this Appendix we present the detailed results on the basis of which we decided to run OCTANE in the subspace of fractional variables (as opposed to the full space), using the expanded octahedron Q^* (as opposed to K^*).

For each problem, we report the details of the acceptable feasible solutions as a 4-tuple

$$(NODE, VALUE, TIME, GAP)$$

where

- *NODE* indicates the node at which the solution was found.
- *VALUE* indicates the objective function value of the solution.
- *TIME* indicates the time at which the solution was found.
- *GAP* indicates the normalized "gap" for the solution by giving its relative deviation from the optimum. If x is the heuristic solution and x^* is the optimal solution, the normalized gap is $\frac{|cx^*-cx|}{|cx^*|}$.

Tables A.1 and A.2 contain the computational results for OCTANE, both for the octahedron and the expanded octahedron, in different spaces of enumeration ("Full Space" and "Fractional Space"). The column "TTime" contains the total time used by the heuristic throughout the entire run.

The tables clearly show that using the fractional space is better than using the full space. Despite the fact that less feasible points are found when using the fractional space, the quality of the points is similar. The time at which points of similar quality are found is usually smaller with the fractional space, although there are a few exceptions (LP4L, LSEU, P0033, P0291 and TSP43). On GAPxx, L152LAV, MISC01, MOD010, P0282 and P0548, the fractional space clearly dominates the full space. The total CPU time is also overwhelmingly in favor of the fractional space.

Comparing the results obtained with the octahedron and the expanded octahedron in the fractional space, we note that, as expected, the total time for the expanded octahedron is larger than the time for the octahedron. The worst ratio of these times is obtained on GAP61 and MISC07 where it approaches 2, very far from the theoretical worst case of n. If we look at the node at which the optimal (or best) solution is found, however, we find that the expanded octahedron is never worse than the octahedron. In terms of the time at which the solution is found, the problems may be roughly divided in two groups. The first one contains the problems for which both heuristics find identical solutions at identical nodes (BM23, LP4L, LSEU, MISC03, MISC07, MOD010, P0201, P0282, P0291, P0548, P2756 and SENTOY) where the expanded octahedron is slightly

slower than the octahedron. On the other problems, however, the latter is clearly superior to the former.

Problem name	K^*			
	Full space		Fractional space	
	Solution	TTime	Solution	TTime
BM23	(2,34,0.02,0.00)	5.16	(0,34,0.42,0.00)	1.93
GAP51	(152,561,20.67,0.003)	13.41	(29,560,3.51,0.005)	4.41
GAP61	(656, 759, 115.53, 0.002)		(208,760,15.32,0.001)	
	(696, 761, 124.65,0.00)	46.38	(448,761,38.03,0.00)	12.27
GAP84	(624,1115,172.57,0.001)		(2008,1117,399.49,0.00)	28.11
	(1272,1116,494.46,0.0008)			
	(1856,1117,620.30,0.00)	98.29		
L152LAV	+++	+++	****	****
LP4L	(4,2967,1.03,0.00)	55.43	(80,2967,52.83,0.00)	11.86
LSEU	(5,1169, 0.06,0.04)		(96,1157,5.59,0.03)	
	(7,1148,0.09,0.02)		(192,1120,9.92 ,0.00)	1.96
	(32,1120,0.62,0.00)	6.85		
MISC01	(200,574.5,16.37,0.02)	4.53	(104, 595.0,6.72,0.06)	1.81
MISC03	(40,3640,10.34,0.08)	11.23	(6,3640,2.54,0.08)	
			(120,3360,13.75,0.00)	3.08
MISC07	****	171.40	****	54.12
MOD008	(0,310, 0.00, 0.009)		(0,308,0.01,0.0003)	
	(7, 307, 0.22, 0.00)	37.30	(1,307,0.86,0.00)	12.31
MOD010	(15,7083,114.36,0.08)	62.37	(2,6553,50.13,0.0007)	5.97
P0033	(3,3089,0.03,0.00)	1.49	(17,3095,0.81,0.001)	
			(96,3089,2.73,0.00)	0.66
P0201	(264,7975,63.65,0.04)		(712,7615,142.81,0.00)	5.36
	(720,7615,180.24,0.00)	20.12		
P0282	(72,264009,49.81,0.02)		(72,259184,24.89,0.002)	
	(128,263902,72.08,0.02)		(112,258643,36.58,0.0008)	5.96
	(208,262550,91.03,0.01)			
	(440,259436,149.98,0.003)	22.32		
P0291	(2,5350,0.06,0.02)		(2,5350.1,2.14,0.02)	
	(7,5311,0.17,0.01)	6.80	(13,5223.7,5.51,0.00)	2.23
P0548	+++	+++	(1632,8744,3201.53,0.01)	764.13
P2756	(232,3174,1773.01)		(304,3227,1533.17,0.03)	
	(560,3134,3951.09,0.003)		(320, 3175,1580.28,0.01)	
	(632,3125,4357.93,0.00)	1370.24	(856,3163,3840.73,0.01)	
			(936,3124,4144.06,0.00)	322.97
PIPEX	(144,788.3,10.83,0.00)	6.28	(208,788.3,7.42,0.00)	2.10
SENTOY	(0,-7761,1.87,0.001)		(0,-7758,2.18,0.001)	
	(160,-7772,12.16,0.00)	3.20	(160,-7772,11.85,0.00)	2.72
TSP43	(0,5967,0.02,0.06)		(3,5633,5.48,0.0005)	
	(1,5625,0.15,0.0008)		(144,5627,47.38,0.001)	
	(3,5620,0.30,0.00)	92.66	(432,5620,166.73,0.00)	6.14

**** No feasible solution found.

+++ Time limit exceeded.

Table A.1: K^* in full space versus the space of fractional variables.

Problem name	Q^*			
	Full space		Fractional space	
	Solution	TTime	Solution	TTime
BM23	(2,34,0.03,0.00)	5.83	(0,34,0.65,0.00)	2.24
GAP51	(0,559,0.89,0.01) (7,560,3.27,0.005) (16,563,8.11,0.00)	 20.12	(29,560,4.36,0.005) (104,562,7.62,0.001)	 5.13
GAP61	(152,755,31.40,0.007) (360,759,61.91,0.002) (432,760,83.21,0.001) (448,761,84.01,0.00)	 80.65	(208,761,19.85,0.00)	21.32
GAP84	(48,1111,20.16,0.01) (1024,1112,233.68,0.004) (1080,1117,249.85,0.00)	 217.90	(24,1110,4.22,0.01) (672,1116,102.11,0.00) (1016,1117,173.06,0.00)	 41.22
L152LAV	+++	+++	(25,4781,53.07,0.01) (28.4749,55.72,0.01) (3704,4722,3631.16,0.00)	 96.23
LP4L	(4,2967,2.91,0.00)	191.36	(80,2967,57.96,0.00)	15.72
LSEU	(5,1169, 0.08,0.04) (7,1148,0.02,0.11) (32,1120,0.68,0.00)	 11.01	(96,1157,6.03,0.03) (192,1120,10.12 ,0.00)	 2.39
MISC01	(17,595.5,3.21,0.06)	5.10	(17,595.5,2.57,0.06)	2.11
MISC03	(6,3640,3.66,0.08) (120,3360,24.79,0.00)	 20.68	(6,3640,2.81,0.08) (120,3360,14.68,0.00)	 3.86
MISC07	****	379.33	****	93.29
MOD008	(0,310,0.00,0.0003) (7,307,0.62,0.00)	 206.71	(0,308,0.01,0.0003) (0,307,1.16,0.00)	 16.43
MOD010	(15,7083,894.18,0.08)	912.73	(2,6553,50.41,0.0007)	6.52
P0033	(3,3089,0.03,0.00)	1.89	(56,3095,1.12,0.001) (120,3089,3.01,0.00)	 0.63
P0201	(264,7975,67.47,0.04) (720,7615,185.37,0.00)	 25.37	(712,7615,146.21,0.00)	8.97
P0282	(72,264009,50.66,0.02) (128,263902,73.13,0.02) (208,262550,92.39,0.01) (440,259436,150.85,0.003)	 27.96	(72,259184,26.16,0.002) (112,258643,37.82,0.0008)	 6.71
P0291	(2,5350.1,0.07,0.02) (7,5311,0.19,0.01)	 8.40	(2,5350.1,2.50,0.02) (13,5223.7,5.95,0.00)	 2.68
P0548	+++	+++	(1632,8744,3288.71,0.01)	948.23
P2756	(232,3174,4090.17,0.01) (560,3134,6240.08,0.003) (632,3125,6643.44,0.00)	 3675.05	(304,3227,1572.84,0.03) (680,3192,3528.40,0.02) (856,3163,3901.25,0.01) (936,3124,4213.06,0.00)	 429.07
PIPEX	(8,789.9,1.24,0.001) (22,788.3,3.36,0.00)	 8.14	(8,789.9,1.15,0.001) (120,788.3,6.01,0.00)	 2.29
SENTOY	(0,-7761,1.94,0.001) (160,-7772,12.81,0.00)	 4.12	(0,-7758,2.20,0.001) (160,-7772,11.76,0.00)	 2.81
TSP43	(0,5625,0.02,0.008) (3,5620,0.37,0.00)	 123.35	(0,5625,4.14,0.00) (152,5620,52.24,0.00)	 6.07

**** No feasible solution found.

+++ Time limit exceeded.

Table A.2: Q^* in full space versus the space of fractional variables.