ORSA Journal on Computing
Vol. 5, No. 2, Spring 1993

0899-1499/93/501-206 $01.25
© 1993 Operations Research Society of America

An Implementation of the Generalized Basis Reduction Algorithm for Integer Programming

WILLIAM COOK / *Bell Communications Research, 445 South Street, Morristown, NJ 07960; Email: bico@bellcore.com*

THOMAS RUTHERFORD / *Department of Economics, University of Western Ontario, London, Ontario N6A 5C2 Canada; Email: tomr@uwovax.uwo.ca*

HERBERT E. SCARF / *Cowles Foundation for Research in Economics, Box 2125, Yale University, New Haven, CT 06520; Email: scarf-herb@cs.yale.edu*

DAVID SHALLCROSS / *Bell Communications Research, 445 South Street, Morristown, NJ 07960; Email: davids@bellcore.com*

(Received: October 1991; revision received: July 1992; accepted: November 1992)

In recent years many advances have been made in solution techniques for specially structured 0-1 integer programming problems. In contrast, very little progress has been made on solving general (mixed) integer problems. This, of course, is not true when viewed from the theoretical side: H.W. Lenstra (1983) made a major breakthrough, obtaining a polynomial-time algorithm when the number of integer variables is fixed. We discuss a practical implementation of a Lenstra-like algorithm, based on the generalized basis reduction method of L. Lovász and H.E. Scarf (1988). This method allows us to avoid the ellipsoidal approximations required in Lenstra's algorithm. We report on the solution of a number of small (but difficult) examples, with up to 100 integer variables. Our computer code uses the linear programming optimizer CPLEX as a subroutine to solve the linear programming problems that arise.

Much of the computational work on integer linear programming has been devoted to solving problems having 0-1 variables. In recent years, the focus has been on the use of problem-specific cutting-plane methods for various classes of 0-1 problems. Some spectacular successes in this area have been achieved by Crowder, Johnson, and Padberg,[5] Grötschel, Jünger and Reinelt,[6] Padberg and Rinaldi,[18] van Roy and Wolsey,[20] and others. In contrast, very little progress has been make on solving general (mixed) integer problems. Indeed, most commercially available codes are based on branch and bound methods that are not particularly well suited for general integer variables. This specialization to algorithms for 0-1 problems is partly justified by the large number of applications in which the variables are restricted to assume the values of 0 and 1. But it is possible that the lack of successful applications of general integer models is due in part to the absence of tools for solving these models.

In 1983, Lenstra[13] made a major breakthrough in the theory of integer programming, obtaining an algorithm with running time bounded by a polynomial in the size of the input when the number of integer variables is fixed. Perhaps more important than the algorithm itself was Lenstra's introduction of results from the geometry of numbers into the study of integer programming. These methods have since been used with great success by Kannan,[8] Kannan and Lovász,[9] Kannan, Lovász, and Scarf,[10] Grötschel, Lovász, and Schrijver[7] and others, in treating a variety of issues in integer programming. Although these results are all of a theoretical nature, it is clear that the geometry of numbers provides insights that can be used in dealing with the practical issue of solving general integer programming problems. The purpose of this paper is to describe a first step in this direction.

We present a practical implementation of a Lenstra-like algorithm, employing the "generalized basis reduction" method of Lovász and Scarf,[15] and report on some computational experience with the algorithm. As one could anticipate, for 0-1 problems our code was not competitive with commercially available codes such as LINDO[14] and OSL.[17] However, for a class of general integer programming problems arising from a network design application (Saniee[21]), our code successfully solved a series of test problems that were not tractable by standard branch and bound techniques.

The paper is organized as follows. In Section 1 and 2 we briefly present the background of the algorithm. In Section

Subject classifications: Programming, integer.
Other key words: Integer programming, geometry of numbers.

3 we describe our implementation and in Section 4 we present some computational results. We assume the reader is familiar with integer programming methods. For a treatment of this area see the books of Nemhauser and Wolsey[16] and Schrijver.[22]

1. Branching on Hyperplanes

In our discussion, we consider the integer programming feasibility problem: Given a system of linear inequalities $Ax \leqslant b$, with A a rational m by n matrix and b a rational m-vector, does there exist an integral vector h satisfying $Ah \leqslant b$? In other words, given a polyhedron $P = \{x : Ax \leqslant b\}$, is $P \cap Z^n$ nonempty? (The set of n-dimensional integral vectors is denoted by Z^n.) In Section 3 we indicate how to deal with an objective function and the possibility of having continuous variables. We will assume throughout the paper that the entries of A and b are rational, so that P is a rational polyhedron.

The key to Lenstra's integer programming algorithm is the following "Flatness Theorem" due to Khinchine.[11]

Theorem 1. *Let $P \subseteq R^n$ be a bounded, rational polyhedron. Then either $P \cap Z^n \neq \emptyset$ or there exists $w \in Z^n \setminus \{0\}$ such that*

$$max\{wx : x \in P\} - min\{wx : x \in P\} \leqslant \gamma_n$$

where γ_n is a constant depending only on the dimension n. ■

This result states that either P contains an integral vector or there exists an integral direction in which P is "flat." To date, the best known upper bound for γ_n is cn^2, where c is a universal constant (Kannan and Lovász[9]).

Lenstra[13] showed that if the dimension n is fixed, then in time polynomial in the size of $Ax \leqslant b$, one of the two alternatives in Theorem 1 can be found for a certain choice of γ_n. This was improved by Grötschel, Lovász, and Schrijver[7] who showed that for $\gamma_n = (n)(n+1)2^{n^2}$ one of the two alternatives can be found in polynomial time even for varying n. In both cases, a polynomial-time algorithm to solve the integer programming feasibility problem can be derived as follows. First, by intersecting P with a large enough box if necessary, we may assume that P is a bounded polyhedron (see Schrijver[22]). Next, applying one of the above algorithms, we either find an integral vector in P or we find a flat direction $d_1 \in Z^n$. In the later case, for each integer t_1 such that $\lceil \min\{d_1 x : x \in P\} \rceil \leqslant t_1 \leqslant \lfloor \max\{d_1 x : x \in P\} \rfloor$ we test if the polyhedron $P \cap \{x : d_1 x = t_1\}$ contains an integral vector. Since d_1 is a flat direction and n is fixed, there are only a constant number of these subproblems. Moreover, it is easy to ensure that the dimension of each of the polyhedra $P \cap \{x : d_1 x = t_1\}$ be less than the dimension of P. So, applying the above procedures to each of these polyhedra, we build a search tree (as in Figure 1) that is at most n deep. This means that the tree has only a constant number of nodes (since n is fixed). It follows that we have a polynomial-time algorithm for solving the original feasibility problem.1

This type of algorithm can be described as "branching on hyperplanes." The flat directions are used to keep the

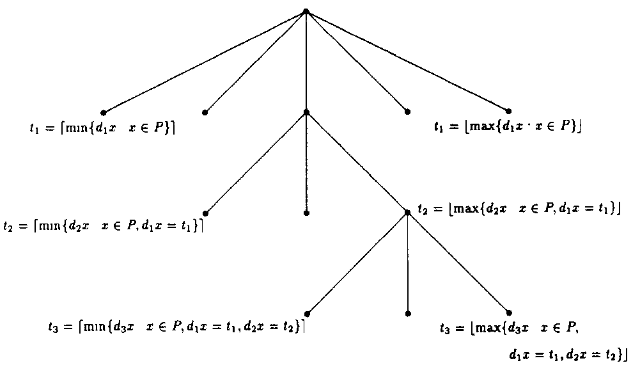
$t_1 = \lceil \min\{d_1 x \;\; x \in P\} \rceil$
$t_1 = \lfloor \max\{d_1 x \cdot x \in P\} \rfloor$
$t_2 = \lceil \min\{d_2 x \;\; x \in P, d_1 x = t_1\} \rceil$
$t_2 = \lfloor \max\{d_2 x \;\; x \in P, d_1 x = t_1\} \rfloor$
$t_3 = \lceil \min\{d_3 x \;\; x \in P, d_1 x = t_1, d_2 x = t_2\} \rceil$
$t_3 = \lfloor \max\{d_3 x \;\; x \in P, d_1 x = t_1, d_2 x = t_2\} \rfloor$

Figure 1. Search tree.

search tree from getting too large. To find these directions, both Lenstra[13] and Grötschel, Lovász, and Schrijver[7] start off by computing a pair of ellipsoids E_1 and E_2 that approximate P in the following sense: $E_1 \subseteq P \subseteq E_2$, E_1 and E_2 are concentric, and E_2 arises by blowing up E_1 by a factor of σ_n, where σ_n is a constant depending only on n. Notice that if we find a flat direction for E_1, then this is also a flat direction for E_2 (and hence also a flat direction for P). So with this ellipsoidal approximation in hand, we can temporarily ignore P and concentrate on the inside ellipsoid E_1. This is precisely what Lenstra and Grötschel, Lovász, and Schrijver do, using a "basis reduction" algorithm of Lenstra, Lenstra, and Lovász[12] to either find a flat direction for E_1 or show that E_1 contains an integral vector.

This ellipsoidal approximation is very convenient, but in practice you do pay a price since a good deal of information is lost in the approximation step. Also, although the pair of ellipsoids can be found in polynomial time (Grötschel, Lovász, and Schrijver[7]), the method makes use of the ellipsoid algorithm for linear programming, which is known not to work well in practice.

In the next section we describe a generalized "basis reduction" algorithm of Lovász and Scarf[15] that we use to find a flat direction by working directly on the polyhedron P. The disadvantage of this method is that it involves considerably more computation than the algorithm of Lenstra, Lenstra, and Lovász.[12] It would be interesting to compare an efficient implementation of Lenstra's algorithm with the results reported in this paper.

2. Generalized Basis Reduction

Let $P = \{x : Ax \leqslant b\}$ be a bounded rational polyhedron. For $w \in R^n$, we define $F(w)$ to be $\max\{wx - wy : Ax \leqslant b, Ay \leqslant b\}$, that is the *width* of P in the direction w (see Figure 2). Notice that the function F is convex, symmetric, and homogeneous of degree 1. To implement the "branching on hyperplanes" strategy, we need to find a nonzero integral vector having a small $F(\cdot)$ value. Our approach will be to find a representation of Z^n that will give us this good direction.

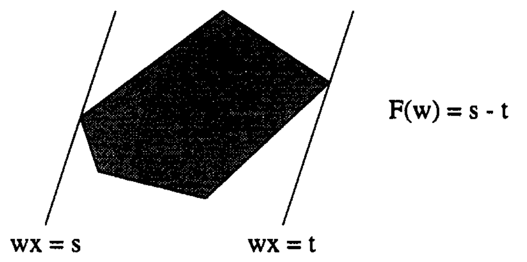

Figure 2. Width of polytope.

A *basis* for the integer lattice Z^n is a collection of linearly independent vectors $b^1, \ldots, b^n$ such that every $w \in Z^n$ can be written as $\lambda_1 b^1 + \cdots + \lambda_n b^n$ for some choice of integers $\lambda_1, \ldots, \lambda_n$. We define a family of functions $F_1(\cdot), \ldots, F_n(\cdot)$ with respect to the polyhedron P and a given basis $b^1, \ldots, b^n$, letting

$$F_i(w) = \min F(w + \alpha_1 b_1 + \cdots + \alpha_{i-1} b^{i-1}) \qquad (1)$$

with the minimum taken over all choices of rational numbers $\alpha_1, \ldots, \alpha_{i-1}$. Note that $F_1(\cdot)$ is just the width function $F(\cdot)$. Using linear programming duality, one can easily show that for each $i = 1, \ldots, n$,

$$F_i(w) = \max\{wx - wy : Ax \leqslant b, Ay \leqslant b, \\ b^1x - b^1y = 0, \ldots, b^{i-1}x - b^{i-1}y = 0\}. \qquad (2)$$

So $F_i(w)$ can, therefore, be computed by solving a single linear programming problem.

To see the connection between the functions $F_i(\cdot)$ and our problem of finding a good direction, consider the following simple result.

Theorem 2. *Suppose $b^1, \ldots, b^n$ is a basis such that $F_1(b^1) \leqslant F_2(b^2) \leqslant \cdots \leqslant F_n(b^n)$. Then b_1 achieves the minimum value of $F(\cdot)$ over all nonzero vectors in Z^n.*

Proof. Let $w \in Z^n \setminus \{0\}$. Since $b^1, \ldots, b^n$ is a basis, we can write $w = \lambda_1 b^1 + \cdots + \lambda_n b^n$ for some integers $\lambda_1, \ldots, \lambda_n$. Let k be the greatest index such that $\lambda_k \neq 0$. We have

$$F_1(w) \geqslant F_k(w) = F_k(\lambda_k b^k) = |\lambda_k| F_k(b^k) \geqslant F_1(b^1).$$

The result follows, since $F_1(\cdot)$ is identical to $F(\cdot)$. ∎

Although we cannot usually achieve the conditions given in Theorem 2, we can find a basis that satisfies the weaker condition

$$F_{i+1}(b^{i+1}) \geqslant ((1/2) - \epsilon) F_i(b^i) \qquad (3)$$

for all $i = 1, \ldots, n-1$, where $0 < \epsilon < 1/2$ is a fixed constant.

To this end, we call a basis $b^1, \ldots, b^n$ *reduced* if the following two conditions hold for all $i = 1, \ldots, n-1$

$$F_i(b^{i+1} + \mu b^i) \geqslant F_i(b^{i+1}) \quad \text{for all integers } \mu \qquad (4)$$

$$F_i(b^{i+1}) \geqslant (1 - \epsilon) F_i(b^i). \qquad (5)$$

These simple conditions on pairs of the basis vectors immediately suggest an algorithm for computing a reduced basis, as we indicate below. But first, note that a reduced basis does indeed satisfy (3) (Lovász and Scarf[15]).

Theorem 3. *Let $b^1, \ldots, b^n$ be a reduced basis. Then* (3) *holds for all $i = 1, \ldots, n-1$.*

Proof. From the definition of $F_i(\cdot)$, we have the identity

$$F_{i+1}(b^{i+1}) = \min\{F_i(b^{i+1} + \alpha b^i) : \text{all rationals } \alpha\}.$$

Now since the rational α that achieves the minimum in the right hand side of this equation is within 1/2 of an integer, we know that the right hand side is at least

$$\min\{F_i(b^{i+1} + \mu b^i) : \text{all integers } \mu\} - (1/2) F_i(b^i).$$

Using the conditions (4) and (5), the result follows. ∎

Combining this with the proof of Theorem 2, we have the following result of Lovász and Scarf.[15]

Theorem 4. *Let $b^1, \ldots, b^n$ be a reduced basis and let λ be the minimum of $F(w)$ over all nonzero $w \in Z^n$. Then $\lambda \geqslant F(b^1)((1/2) - \epsilon)^{n-1}$.* ∎

It follows that a reduced basis gives us a direction that is not too far away from the minimum width of the polytope P.

To aid the discussion in the next section, we state the *generalized basis reduction* algorithm that follows from the conditions (4), (5) given above. (The "generalized" refers to the fact that this algorithm generalizes the basis reduction method given in Lenstra, Lenstra, and Lovász[12].) The algorithm starts off with a basis $b^1, \ldots, b^n$ (for example, the *standard basis* where each b^i is the ith unit vector) and moves through a sequence of bases as follows: Starting with $i = 1$ and continuing until $i = n$, carry out the two steps

- Replace b^{i+1} by $b^{i+1} + \mu b^i$, with μ the integer that minimizes $F_i(b^{i+1} + \mu b^i)$.
- If $F_i(b^{i+1}) < (1 - \epsilon) F_i(b^i)$, swap b^i and b^{i+1} and set i to the maximum of 1 and $i - 1$. Otherwise, set i to $i + 1$.

Lovász and Scarf[15] show that this algorithm runs in polynomial time for fixed n. (It is not known whether the running time is polynomial for varying n.) For a further discussion of reduced bases we refer to the reader to the paper of Lovász and Scarf[15].

3. Implementation

Carrying out the generalized basis reduction algorithm involves the solution of many linear programming problems. Several important observations help to cut down on this computational effort.

To start off, how do we find the integer μ^* that minimizes $F_i(b^{i+1} + \mu b^i)$? This could be done with a line search, but there is a more direct method. Since $F_{i+1}(b^{i+1}) = \min\{F_i(b^{i+1} + \alpha b^i) : \text{all rationals } \alpha\}$, it follows that the rational α^* that minimizes $F_i(b^{i+1} + \alpha b^i)$ is the dual variable associated with the equation $b^i x - b^i y = 0$ in the linear programming problem used in the calculation of $F_{i+1}(b^{i+1})$. Since $F_i(b^{i+1} + \alpha b^i)$ is a convex function in α, we know that μ^* is either $\lceil \alpha^* \rceil$ or $\lfloor \alpha^* \rfloor$.

Using the above approach, at level i of the basis reduction algorithm we will compute the functions

$$F_{i+1}(b^{i+1}), F_i(b^{i+1} + \mu_i b^i), F_i(b^i) \tag{6}$$

where μ_i is the integer that minimizes $F_i(b^{i+1} + \mu b^i)$, together with α_i, the dual variable associated with the equation $b_i x - b^i y = 0$ in the linear programming problem for $F_{i+1}(b^{i+1})$. Finding these data in direct manner will require the solution of either two or four linear programming problems, depending on whether or not α_i^* is an integer. Some of this work can be saved, however, by simply avoiding the solution of problems that we have already solved, as we indicate below.

Suppose that we are at level i and are about to decrease the level to $i - 1$. In this transition we first replace b^{i+1} by $b^{i+1} + \mu_i b^i$ and swap this latter vector with b^i, obtaining the basis

$$b^1, b^2, \dots, b^{i-1}, b^{i+1} + \mu_i b^i, b^i \cdots b^n.$$

The first of the three linear problems that we must solve at this level involves the evaluation of $F_i(b^{i+1} + \mu_i b^i)$, and the third the evaluation of $F_{i-1}(b^{i-1})$.

The first observation to make is that $F_{i-1}(b^{i-1})$ has already been evaluated at the last time that we were at level $i - 1$, since we have not changed the vectors b^1, $b^2, \dots, b^{i-1}$ in any of the basis reduction steps at levels greater than $i - 1$. Therefore, all that we need in order to evaluate this function is a list of its past values.

Secondly, the function $F_i(b^{i+1} + \mu_i b^i)$ is precisely the same as the middle function in (6) at level i, and therefore need not be revaluated. We can also obtain α_{i-1} from this same evaluation. (Note that if we did not have to evaluate the middle function at level i, because α_i was an integer, then α_{i-1} is the dual variable associated with the constraint $b^{i-1}x - b^{i-1}y = 0$ in the evaluation of $F_{i+1}(b^{i+1})$.)

So, if α_{i-1} is an integer, which it is in many cases, then there are no new linear programming problems to solve at this level. If α_{i-1} is not an integer, then we need to solve two linear programming problems, in order to get the middle function in (6). More generally suppose that the dual variables $\alpha_{i-1}, \dots, \alpha_{i-k}$ are all integral. Then the first function in (6) stays constant, and the middle function is not evaluated, for the next k iterations, as long as the index is decreasing.

There is also a savings that can be made if we move from level i to level $i + 1$. In this case, the new basis is

$$b^1, \dots, b^{i-1}, b^i, b^{i+1} + \mu_i b^i, \dots, b^n.$$

Now the last of the functions in (6) to be evaluated at level $i + 1$ is $F_{i+1}(b^{i+1} + \mu_i b_i)$. But this value is the same as $F_{i+1}(b^{i+1})$, which we have already evaluated at the previous level. Again, this can be continued for any sequence of increasing indices.

In our implementation, rather than working with a general basis $b^1, \dots, b^n$, we perform unimodular transformations of the space so that our basis is always the standard basis (with b^i the ith unit vector). This makes the computer code somewhat simpler, since we know beforehand the form of the equations $b^i x - b^i y = 0$ that must be added to the linear programming problems. Note, however, that we must maintain the inverse transformation, B^{-1}, to bring any integral vector we find back to the original coordinate system. To describe the necessary transformations, let $a^1, \dots, a^n$ denote the columns of the matrix A. Then replacing b^{i+1} by $b^{i+1} + \mu b^i$ is accomplished by replacing a^i by $a^i - \mu a^{i+1}$, and swapping b^i and b^{i+1} is accomplished by swapping a^i and a^{i+1}. To update our inverse transformation, we perform these same operations on the columns of B^{-1}. (A potential disadvantage of this method is that it might lead to a greater increase in the number of non-zeros than had we appended the reduced basis explicitly. We have not, however, encountered extensive fill-in with any of the test problems which we have examined.)

The basis reduction algorithm is the core of our integer programming method. As we outlined in Section 1, our code proceeds as follows (keeping in mind that we always work with the standard basis). We have a recursive function $SEARCH(k, t_1, t_2, \dots, t_{k-1})$, where k and t_1, $t_2, \dots, t_{k-1}$ are integer inputs. When $SEARCH(k, t_1, t_2, \dots, t_{k-1})$ is called, the first $k - 1$ variables have been fixed at the integer values $t_1, t_2, \dots, t_{k-1}$, and the function will search for integer values for the remaining variables. If k happens to be equal to $n + 1$, then the function simply records the integral solution $(t_1, \dots, t_n)$ (after transforming it back to the original coordinate system) and terminates the program. Otherwise, the variable x_k is maximized and minimized over the current polyhedron (the transformed original polyhedron, intersected with the hyperplanes $x_1 = t_1, \dots, x_{k-1} = t_{k-1}$), giving us two numbers, u and l (the optimal values of the two linear programming problems). If $\lfloor u \rfloor < \lceil l \rceil$, then we know that the current polyhedron contains no integral vectors, so we return 0. If $\lfloor u \rfloor = \lceil l \rceil$, then we fix x_k at this value, and return the output of $SEARCH(k + 1, t_1, t_2, \dots, t_{k-1}, \lfloor u \rfloor)$. If $\lfloor u \rfloor > \lceil l \rceil$, then there is more than one possibility for x_k. So we call the basis reduction routine (with the parameter k, that tells the routine only to perform the basis reduction on the basis vectors $b^k, \dots, b^n$), to transform the problem in an attempt to cut down on the number of possible values for x_k. We than recompute u and l. If $\lfloor u \rfloor < \lceil l \rceil$ or $\lfloor u \rfloor = \lceil l \rceil$, we proceed as above. Otherwise, for each integer t_k between $\lceil l \rceil$ and $\lfloor u \rfloor$ we set $x_k = t_k$ and call $SEARCH(k + 1, t_1, t_2, \dots, t_{k-1}, t_k)$. If each of these calls returns 0, then $SEARCH(k, t_1, t_2, \dots, t_{k-1})$ also returns the value 0.

This procedure solves the integer programming feasibility problem; it either finds an integral solution (and terminates) or $SEARCH(1, \emptyset)$ returns the value 0, in which case we can conclude that the polyhedron contains no integral vectors. But since many practical problems will have an associated objective function that must be maximized, that is, they will have the form $\max\{wx : Ax < b,\ x \text{ integral}\}$, we need to deal with this possibility in our code.

One straightforward method is to simply add the constraint $wx \leqslant T$ to our system $Ax \leqslant b$, and use binary search on the value of T (solving the feasibility problem at each step) to obtain the optimal solution. This has the disadvantage that we repeatedly search the same region of the polyhedron on each call to the feasibility procedure.

Another approach is to continue the search when we find an integral solution $(x_1, \ldots, x_n)$ by raising the value of T to a small amount greater than the objective value $w_1 x_1 + \cdots + w_n x_n$ (so we limit the further search to integral solutions having greater objective value). This allows us to avoid revisiting parts of the polyhedron, but its performance depends on having a good initial value of T. It is easy to adapt either method to stop whenever we are within a certain percentage, p, of optimality. In the latter case, rather than increasing T by a small amount, we can increase by $(p/100)|T|$.

Finally, we need to describe how to handle mixed integer programming problems. This is particularly simple: the basis reduction is carried out only on the vectors corresponding to integer variables. Thus, the only effect of the continuous variables is to increase the sizes of the linear programming problems that we need to solve.

4. Computational Results

Our computer code is written in the C programming language and uses the linear programming optimizer CPLEX[4] to solve the linear programming problems that arise. Extensive use is made of CPLEX's Callable Library, to modify the constraint matrix A and to add and delete the equations $b'x - b'y = 0$ during the course of the algorithm. Also, since the linear programming problems that must be solved are closely related to one another, we make use of CPLEX's ability to begin with the optimal (LP) basis from one problem to "jump start" the solution of the next problem.

Due to the complexity of the generalized basis reduction algorithm, our code is suitable only for problems having not too many more than 100 integer variables (and any number of continuous variables). Moreover, since even a single run of the basis reduction algorithm involves the solution of a large number of linear programming problems, our code should be used only if simple branch and bound techniques have difficulty on the particular instance. We report on a set of eight such problems in the tables below. The first four were provided by Ellis Johnson, and are part of a set of problems collected at IBM's Watson Research Center. The remaining four problems are instances of a network design problem at Bellcore, described by Saniee,[21] and studied in Bienstock,[1,2] and Pochet and Wolsey.[19] Six of the eight problems (lseu, p0033, and the four bell problems) are contained in the MIPLIB library of test problems collected by Bixby, Boyd, and Indovina.[3] (These problems can be obtained by sending email to *softlib@rice.edu* with the message *send catalog*.)

Some information about the problems is presented in Table I. The "Variables" column is the total number of variables in the problem and the "Integer" column is the number of integer variables. All eight instances are formulated as minimization problems, and all problems contain variables that can take on general integer values. (The first four problems were originally presented as 0–1 problems.)

The solution times for four versions of our code are given in Table II. The first variant, "Short," stops the basis reduction whenever it returns to level 1. This cuts down on the running time of the basis reduction algorithm, but may produce inferior branching hyperplanes. In the second version, "Full," we allow the basis reduction to run to completion. In "Fractional," we allow the basis reduction to run to completion, but we only carry out the basis reduction algorithm on the vectors corresponding to the fractional values in the current linear programming solution. In each of these three instances, we search for an optimal solution with the straight search method described in the previous section (that is, we do not use binary search). In the fourth version, "Binary Search," we again use "Short" basis reduction, but, in this case, binary search on the objective value is used until we reach a polyhedron that contains no integral vectors, then we switch to a straight search from the objective value of the best solution we have found up to that point.

The final column of Table II reports the solution times of a straightforward, depth-first-search branch and bound code we implemented for comparison purposes. (Again, this code uses CPLEX as a linear programming solver.) In two instances (bellnet3b and bellnet4), we were unable to solve the problems with "BRANCH." The times reported for these instances are the points where we stopped the code.

Each test reported in Table II began with the bound listed in the "Bound" column. For the first four problems, these bounds are the objective values of the solutions found by an integer programming heuristic that we implemented. (The heuristic failed to find a solution to problem ip.5). For the four network problems, starting with the optimal value of the linear programming relaxation, we tried increasing the bounds in increments of 100000 (or 10000 in the case of bellnet3a) until a feasible solution was found. (When the bounds were set too low, the infeasibility of the problem was detected very rapidly.)

As can be seen, our basis reduction code ("BIRCH") was competitive with BRANCH, and was significantly better on the network design problems. One factor that distinguishes these problems from the first four is that the optimal mixed-integer solutions take on values significantly greater than 1 (which is not the case in the other problems, perhaps due to the fact that they were originally modeled with 0–1 variables).

The advantage of using basis reduction is shown more clearly in Table III, where the number of nodes in the search trees for BIRCH (with the "Short" basis reduction) and BRANCH are presented. In this case, BIRCH was

Table I. Test Problems

Problem	Variables	Integer	Rows	LP Opt	MIP Opt
lseu	89	89	28	662.97	953
p0033	33	33	16	1734.18	2455
ip.5	353	31	272	2690.01	2691.69
ip.6	74	74	38	−638.57	−540
bell3a	133	71	124	862578.64	878430.32
bell3b	133	71	124	11404143.86	11786160.62
bell4	117	64	106	17984775.91	18541484.20
bell5	104	58	92	8608417.95	8966406.49

Table II. CPU Seconds on an IBM R6000 Model 530

Problem	Bound	Short	Full	Fractional	Binary Search	BRANCH
lseu	1153	7180.6	9274.2	5027.6	10605.7	393.8
p0033	3089	79.9	95.0	42.6	118.0	387.1
ip.5	none	12796.4	29662.1	6213.9	2244.9	1743.3
ip.6	493	3312.6	3441.6	4157.7	4850.4	1409.4
bell3a	880000	66.8	66.9	346.6	100.1	7034.0
bell3b	11800000	6723.8	8244.7	7486.4	7052.1	+50 hours
bell4	18600000	6880.9	7041.2	39868.8	7051.2	+25 hours
bell5	9000000	1346.4	1376.2	2454.7	1516.9	1698.1

Table III. Node Counts

Problem	BIRCH	BRANCH	Best Branch
lseu	10797	45765	25480
p0033	375	89541	6591
ip.5	3066	28279	21725
ip.6	2910	113849	9036
bellnet3a	113	248011	103748
bellnet3b	5312	+750000	265096
bellnet4	15003	+500000	******
bellnet5	1577	111845	44166

significantly better than BRANCH in all instances. Moreover, BIRCH also has lower node counts than a "Best Branch" branch and bound algorithm that we implemented. In this algorithm, for each fractional value x_i^* in the current linear programming relaxation we compute the width of the current polyhedron in the direction of x_i, and we branch on the variable having minimum width. This is a "best possible" strategy for getting a small search tree, when we restrict our branching hyperplanes to the unit directions. (This algorithm is very time consuming, and is only interesting from the node count perspective.)

We should remark that the poor performance of BRANCH on the network design problems is not only due to its simple design. The much more sophisticated commercial branch and bound codes, LINDO[14] and OSL Version 1.1[17] failed to solve any of the four test instances. In the case of LINDO, attempts to solve each of the four problems terminated after 40 hours on a VAX 780 without finding a feasible solution. Using OSL's integer programming solver, each run exceeded the 220 megabyte allocated workspace (and terminated) before finding a feasible solution.

The success of BIRCH on the network problems, as well as the improved node counts on the first four problems, suggests that some variant of generalized basis reduction may be a useful tool in approaching classes of difficult general integer programming problems.

Acknowledgments

We would like to thank Imre Bárány, Ravi Kannan, and Lásló Lovász for many useful conversations and suggestions. We would like to thank Ellis Johnson, George Nemhauser, Iraj Saniee and Laurence Wolsey for providing us with sets of test problems, and to thank David Applegate for providing an implementation of gray codes that we used in our integer programming heuristic, Bob Bixby for his help in using the CPLEX Callable Library, Judith Bender for her work on an early version of the code, and Dicky Yan for carrying out the test of LINDO. This research has been supported by the Program on Discrete Mathematics at the Cowles Foundation. Research of the second author supported under operating grant T306A1 from the Canadian Natural Science Research Council, and the third author supported under National Science Foundation grant SES-8807167.

References

1. D. BIENSTOCK, 1992. A Lot-Sizing Problem in Trees, Related to Network Design, in E. Balas, G. Cornuéjols, and R. Kannan (eds.), *Integer Programming and Combinatorial Optimization—Proceedings of a Conference*, Carnegie Mellon University, Carnegie Mellon University Press, Pittsburgh, pp. 421–434.
2. D. BIENSTOCK, 1992. Computational Experience with an Effective Heuristic for Some Capacity Expansion Problems in Local Access Networks, Technical Report No. 17, Department of Industrial Engineering and Operations Research, Columbia University, New York.
3. R.E. BIXBY, E.A. BOYD and R. INDOVINA, 1992. MIPLIB: A Test Set of Mixed Integer Programming Problems, *SIAM News 25.2*, 16.
4. CPLEX, 1992. Cplex Optimization Incorporated, Incline Village, NV.
5. H. CROWDER, E.L. JOHNSON and M. PADBERG, 1983. Solving Large-Scale Zero-One Linear Programming Problems, *Operations Research 31*, 803–834.
6. M. GRÖTSCHEL, M. JÜNGER and G. REINELT, 1984. A Cutting Plane Algorithm for the Linear Ordering Problem, *Operations Research 32*, 1195–1220.
7. M. GRÖTSHCEL, L. LOVÁSZ and A. SCHRIJVER, 1988. *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
8. R. KANNAN, Minkowski's Convex Body Theorem and Integer Programming, *Mathematics of Operations Research 12*, 415–440.
9. R. KANNAN and L. LOVÁSZ, Covering Minima and Lattice-Point-Free Convex Bodies, *Annals of Mathematics 128*, 577–602.
10. R. KANNAN, L. LOVÁSZ and H.E. SCARF, 1990. The Shapes of Polyhedra, *Mathematics of Operations Research 15*, 364–380.
11. A. KHINCHINE, 1948. A Quantitative Formulation of Kronecker's Theory of Approximation (in Russian) Izvestiya Akademii Nauk SSR Seriya Matematika Akad. Nauk. SSSR, Ser. Mat. *12*, 113–122.

12. A.K. LENSTRA, H.W. LENSTRA and L. LOVÁSZ, 1992. Factoring Polynomials with Rational Coefficients, *Mathematics Annalen 261*, 513–534.
13. H.W. LENSTRA, 1983. Integer Programming with a Fixed Number of Variables, *Mathematics of Operations Research 8*, 538–548.
14. LINDO, 1992. Lindo Systems Incorporated, Chicago, IL.
15. L. LOVÁSZ and H.E. SCARF, 1992. The Generalized Basis Reduction Algorithm, to appear.
16. G.L. NEMHAUSER and L.A. WOLSEY, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.
17. OSL, 1992. International Business Machines (IBM), Dallas, TX.
18. M. PADBERG and G. RINALDI, 1991. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems," *SIAM Review 33*, 60–100.
19. Y. POCHET and L. WOLSEY, 1992. Network Design with Divisible Capacities: Aggregated Flow and Knapsack Subproblems, in E. Balas, G. Cornuejols, and R. Kannan (eds.), *Integer Programming and Combinatorial Optimization—Proceedings of a Conference*, Carnegie Mellon University, Carnegie Mellon University Press, Pittsburgh, pp. 150–164.
20. T. J. VAN ROY and L.A. WOLSEY, 1987. Solving Mixed Integer Programming Problems Using Automatic Reformulation, *Operations Research 35*, 45–57.
21. I. SANIEE, 1991. Economic Capacity Expansion of the Local Access Network," Technical Memorandum TM-TSV-018565, Bell Communications Research.
22. A. SCHRIJVER, 1986. *Theory of Linear and Integer Programming*, Wiley, Chichester.