Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

DANIEL E. STEFFY[1]
KATI WOLTER[2]

Valid Linear Programming Bounds for Exact Mixed-Integer Programming

[1] Zuse Institute Berlin, Germany. Research supported by NSF Grant CMMI-0726370, ONR Grant N00014-08-1-1104.
[2] Zuse Institute Berlin, Germany. Research funded by DFG Priority Program 1307 “Algorithm Engineering”.

Valid Linear Programming Bounds for Exact Mixed-Integer Programming

Daniel E. Steffy, Kati Wolter
Department of Optimization, Zuse Institute Berlin,
Takustr. 7, 14195 Berlin, Germany, {steffy@zib.de, wolter@zib.de}

Fast computation of valid linear programming (LP) bounds serves as an important subroutine for solving mixed-integer programming problems exactly. We introduce a new method for computing valid LP bounds designed for this application. The algorithm corrects approximate LP dual solutions to be exactly feasible, giving a valid bound. Solutions are repaired by performing a projection and a shift to ensure all constraints are satisfied; bound computations are accelerated by reusing structural information through the branch-and-bound tree. We demonstrate this method to be widely applicable and faster than solving a sequence of exact LPs. Several variations of the algorithm are described and computationally evaluated in an exact branch-and-bound algorithm within the mixed-integer programming framework SCIP.

Key words: mixed-integer programming; exact computation
History:

---

1. Introduction

Software to solve mixed-integer programming (MIP) problems is widely used in both industrial and academic settings. However, due to the use of floating-point arithmetic, most software packages are susceptible to numerical mistakes which can lead to incorrect results. While a degree of numerical error is often tolerated by users in some settings, there are a number of applications where truly correct and exact solutions are desirable or necessary. Such areas include the use of MIP models to establish theoretical results, to verify the correctness of VLSI chip designs, or to determine winners for combinatorial auctions; additional examples are given in (Cook et al., 2011; Steffy, 2011).

Implementing software to solve LPs and MIPs entirely in exact arithmetic can result in a considerable slowdown. Recent work has focused on developing efficient methods to solve LPs problems exactly over the rational numbers using a mix of floating-point and exact computation (Applegate et al., 2007a; Dhiflaoui et al., 2003; Espinoza, 2006; Koch, 2004; Kwappik,

1998). A solver based on these ideas is implemented and studied by Applegate et al. (2007a) as QSOPT_EX (Applegate et al., 2007b). These methods exploit the fact that even in the presence of some numerical errors, floating-point LP solvers are often able to find an optimal or near optimal LP basis. Once an optimal LP basis is identified, the exact rational solution can be computed and verified without requiring that the earlier steps of the algorithm were performed exactly.

In (Applegate et al., 2007a) an exact rational MIP solver based on QSOPT_EX was tested in which an exact branch-and-bound tree was maintained and each LP encountered was solved exactly. While QSOPT_EX was only moderately slower than the floating-point LP solvers, the exact MIP code experienced a more significant slowdown when compared to commercial solvers. In order to improve running times for exact MIP it has been observed that solving the LP relaxation at each node exactly is not always necessary, as long as valid LP bounds can be computed. This idea is discussed by Applegate et al. (2006) and Neumaier and Shcherbina (2004). A recently developed exact rational MIP solver, described by Cook et al. (2011), uses a combination of exact rational arithmetic and safe floating-point computation.

In Section 2, we describe some known methods for generating valid bounds for LP problems. In Section 3, we describe a new algorithm for generating valid LP dual bounds, which we will refer to as the project-and-shift method. A description of our computational experiments is presented in Section 4 and conclusions and future work are discussed in Section 5.

2. Previous Work

The most straightforward way of computing valid LP bounds at nodes of a branch-and-bound tree is to solve each LP relaxation exactly. The node LPs in a branch-and-bound tree are typically solved by the dual simplex algorithm which can be warm started with an optimal basis from the parent node; reoptimization can often be accomplished with a small number of simplex pivots. This type of warm start can also be used when solving node LPs exactly. However, computing exact LP solutions in this way may still be much slower than the floating-point LP solver. Even in the case when the exact LP solver quickly determines the optimal basis by performing additional pivots in floating-point arithmetic, it would still compute an exact solution and verify its optimality at that node, which can be time consuming. The cost associated with computing numerous node LP solutions exactly is

an explanation for why the exact MIP solver tested in (Applegate et al., 2007a) experienced a greater relative slowdown than their exact LP solver, when compared to floating-point codes.

Despite the possible disadvantages of solving an exact LP at every node, it is important to recognize that an exact LP solver will be a necessary component of an exact MIP solver and is used to compute exact primal solutions. An exact LP solver also has the advantage that it will provide the tightest valid LP bound at any node of the branch-and-bound tree.

Any feasible dual solution gives a valid bound on the primal LP objective value. In many cases an approximate dual solution can be corrected to generate a valid bound in this manner. If all primal variables have finite upper and lower bounds then this structure allows any approximate dual solution to be corrected by adjusting the dual variables corresponding to the primal variable bounds. This idea was used to compute valid dual LP bounds within the Concorde software package which is designed to solve Traveling Salesman Problem (TSP) instances by branch-and-cut where each variable is bounded by zero and one (Applegate et al., 2006). Neumaier and Shcherbina (2004) described this procedure more generally for MIPs having finite upper and lower bounds on all primal variables. Consider the following primal dual pair of LPs:

Primal:	**Dual:**
$\max \ c^T x$	$\min \ b^T y - l^T z_l + u^T z_u$
$\text{s.t.} \ Ax \leq b$	$\text{s.t.} \ A^T y \ - I z_l \ + I z_u \ = c$
$l \leq x \leq u$	$y, z_l, z_u \geq 0$

Any approximate dual solution $\tilde{y}, \tilde{z}_l, \tilde{z}_u \geq 0$ can be corrected to be exactly dual feasible by increasing z_l, z_u. If $r = c - A^T\tilde{y} + I\tilde{z}_l - I\tilde{z}_u$ is the error of the approximate solution, then a feasible solution is given by: $(y, z_l, z_u) = (\tilde{y}, \tilde{z}_l + r^+, \tilde{z}_u + r^-)$. Where $r_i^+ = \max(r_i, 0)$ and $r_i^- = \max(-r_i, 0)$. This gives $b^T y - l^T z_l + u^T z_u$ as a valid upper bound on the primal objective. Since this dual bounding method corrects approximate dual solutions using the dual variables coming from primal bound constraints we will call it the *primal-bound-shift method*. The difference between the bound and the objective value of the approximate dual solution will be small if the approximate dual solution does not violate the constraints by a large amount and the bounds l, u on the primal variables are not large.

Proposition 2.1. *Let $\tilde{y}, \tilde{z}_l, \tilde{z}_u \geq 0$ be an approximate dual solution, with cost $b^T\tilde{y} - l^T\tilde{z}_l + u^T\tilde{z}_u$, then the bound computed by the primal-bound-shift method described above will be $-l^T r^+ + u^T r^-$ larger than the cost of the approximate dual solution (if computed exactly).*

Proof. Subtracting the objective value of the approximate solution from the objective value of the corrected solution gives: $(b^T y - l^T z_l + u^T z_u) - (b^T\tilde{y} - l^T\tilde{z}_l + u^T\tilde{z}_u) = -l^T r^+ + u^T r^-$. □

Neumaier and Shcherbina observed that exact precision arithmetic can be entirely avoided when computing r and the bound by using floating-point computation and interval arithmetic (or directed rounding if the problem is described in floating-point representable numbers). The strength and simplicity of computing this bound suggests that it will be an excellent choice when tight primal variable bounds are available. The drawback is that if some variable bounds are very large or missing then it could produce weak or infinite bounds. We found that in our test set, 31 out of 59 problems were missing at least some variable bounds, which could lead to failure of this method.

Some recent studies (Althaus and Dumitriu, 2009; Jansson, 2004; Keil and Jansson, 2006) have looked at solving or detecting feasibility of LPs using interval methods. The methods presented in these articles are more general and sophisticated than the primal-bound-shift method. Althaus and Dumitriu (2009) describe an algorithm to certify feasibility and produce valid bounds for LPs. Their algorithm identifies the implied equalities of an LP and then, using safe interval methods, corrects the interval solution to satisfy all of the constraints by shifting it toward the relative interior of the polyhedron. They implemented a version of the algorithm to certify feasibility of problems and experienced a high rate of success. A variant of their algorithm for computing valid LP bounds is also described. Their method does not require special assumptions on the problem structure and most computations can be performed using fast interval methods. However, it requires the solution of an auxiliary problem to identify the implied equalities each time a bound is computed and can potentially fail when numerical problems are encountered.

3. Project-and-Shift

The methods described in the previous section determine a valid dual solution, or an interval containing a valid dual solution by correcting an approximate dual solution. Similarly, the method presented in this section will generate valid bounds by repairing approximate dual

solutions to be exactly feasible. An approximate solution is projected to satisfy all of the equality constraints and then shifted toward the feasible region to satisfy all of the remaining inequalities. We will not require upper and lower bounds on primal variables. However, we will impose some conditions on the problem structure in order to effectively reuse information throughout the branch-and-bound tree when solving a MIP.

3.1 Basic Idea

The bounding method is defined in terms of the dual problem, where we use the following notation for the root and node LP relaxations of the MIP, where we have $A \in \mathbb{Q}^{m \times n}$, $c, x \in \mathbb{Q}^n$, $b, b', y \in \mathbb{Q}^m$, $\bar{A} \in \mathbb{Q}^{\bar{m} \times n}$ and $\bar{b}, z \in \mathbb{Q}^{\bar{m}}$.

Root Primal:	**Root Dual:**	**Node Primal:**	**Node Dual:**
$\max \; c^T x$	$\min \; b^T y$	$\max \; c^T x$	$\min \; b'^T y + \bar{b}^T z$
$\text{s.t.} \; Ax \leq b$	$\text{s.t.} \; A^T y = c$	$\text{s.t.} \; Ax \leq b'$	$\text{s.t.} \; A^T y + \bar{A}^T z = c$
	$y \geq 0$	$\bar{A}x \leq \bar{b}$	$y, z \geq 0$

In this notation, modification of the primal variable bounds caused by branching would correspond to lowering components of b, or to introducing new inequalities if the bound was previously infinite. We can assume that the LP relaxation at any node has $b' \leq b$. Adding a cutting plane corresponds to adding a new inequality to the primal problem and thus a new column to the dual problem. A solution feasible for the root node dual LP will be feasible for all of the node dual problems in the branch-and-bound tree by setting the additional components z to zero.

Algorithm 1 LP bound by dual correction (single LP version)

Input: Dual constraints $A^T y = c, y \geq 0$, approximate solution $\tilde{y}$
Determine implied equalities of dual polyhedron
Compute relative interior point y^* of dual polyhedron
Fix implied zero components of $\tilde{y}$ to zero
Project $\tilde{y}$ to satisfy $A^T \tilde{y} = c$
Set y equal to a convex combination of $\tilde{y}$ and y^* such that $y \geq 0$
Return: Dual bound $b^T y$

Algorithm 1 is a simplified description of the project-and-shift algorithm as it would be applied to a single LP. It is described using the notation of the root node as given above. As long as the dual LP is feasible this algorithm will always produce a valid bound. It makes use of an approximate dual solution and corrects it to be exactly feasible. The

approximate solution is projected into the affine hull of the feasible region in order to satisfy all equality constraints and then shifted toward the relative interior of the polyhedron to satisfy all inequality constraints. The general idea of Algorithm 1 is also the basis for the work of Althaus and Dumitriu (2009) who use interval arithmetic to reduce the use of exact computation.

In principle, Algorithm 1 could be applied to generate a valid LP bound at each node of the branch-and-bound tree when solving a MIP, but the operations required could be quite expensive; the operations of determining the implied equality constraints of the polyhedron and finding a relative interior solution could be more difficult than solving the exact LP in the first place.

We now adopt this procedure to work efficiently in a MIP setting by performing the most expensive computations only once at the root node of the branch-and-bound tree and reusing the structural information in order to decrease the work performed for each bound computation. The algorithm we describe has two components. A setup phase that must be performed once at the root node, and a bound computation operation that can be called at nodes of the branch-and-bound tree. We make the assumption that the matrix A^T has full row rank and that there are no implied equalities. We will later demonstrate that this assumption is often satisfied on a large set of real-world problems and is more general than the assumption that all primal upper and lower bounds are finite.

In the setup phase, given as Algorithm 2, we choose a submatrix A_S^T of A^T by selecting a subset S of the columns that span $\mathbb{R}^n$. This subset S could be chosen to be all of the columns. Then an LU factorization of A_S^T is computed. Finally a corrector point y^* is computed such that it is dual feasible and that it has strictly positive values in all components corresponding to columns in S. We will define such a point to be an *S-interior point*.

Algorithm 2 Project-and-shift: setup phase

Input: Root dual constraints $A^T y = c, y \geq 0$
Choose subset S of columns of A^T spanning $\mathbb{R}^n$
Compute exact LU factorization of submatrix A_S^T induced by S
Compute exact S-interior dual solution y^*
Return: S, LU, y^*

The actual node bound computation is given as Algorithm 3. An approximate dual solution, $\tilde{y}, \tilde{z} \geq 0$, is corrected to be exactly feasible. First, the violation of the equality constraints r is computed and an adjustment correcting this violation $w \in \mathbb{R}^m$ is calculated

using the LU factorization found in the setup phase; w projects the approximate solution to satisfy the equality constraints. Note that the approximate dual solution is preconditioned to be non-negative. In the case that some components are slightly negative due to numerical errors, those components can be set to zero.

Algorithm 3 Project-and-shift: node bound computation

Input: Node dual constraints $A^T y + \bar{A}^T z = c,\ y, z \geq 0$, approximate dual sol. $\tilde{y}, \tilde{z} \geq 0$
Compute error in equality constraints $r = c - A^T \tilde{y} - \bar{A}^T \tilde{z}$
Solve $A^T w = r$ using precomputed LU factorization of A_S^T
Choose smallest $\lambda \in [0, 1]$ such that $(y, z) = (1 - \lambda)(\tilde{y} + w, \tilde{z}) + \lambda(y^*, 0)$ is non-negative
Return: Dual bound $b'^T y + \bar{b}^T z$

Choosing a value of $\lambda \in [0, 1]$ such that $(y, z) = (1 - \lambda)(\tilde{y} + w, \tilde{z}) + \lambda(y^*, 0) \geq 0$ is always possible. This is because all components of $\tilde{y}, \tilde{z}$ are non-negative, with the only negativity in $(\tilde{y} + w, \tilde{z})$ coming from w whose support is a subset of the columns in S. By definition y^* is strictly positive in all components of S. Furthermore, feasibility of (y, z) is guaranteed because it is a convex combination of two solutions to the equality constraints of the system. This explains the use of our assumptions: the assumption that the columns of S span $\mathbb{R}^n$ implies that the LU factorization of A_S^T can be used to correct any violation r; the assumption that there are no implied equalities ensures the existence of an S-interior point y^* that can be used to correct any negativity appearing in w.

We will refer to the use of Algorithms 2 and 3 together within a MIP branch-and-bound tree as the *project-and-shift method*. Algorithm 1 gave a simplified description of this method as it would be applied to a single LP.

The setup phase will require solving one or two exact LPs, that will be described in Section 3.4, and computing an exact LU factorization. The node bound computations will require considerably less computation, the most expensive part being the back-solve of a system of equations that is done with a precomputed LU factorization. The project-and-shift method may be relatively slow if used to compute a single LP bound, but in a branch-and-bound tree it could prove more effective, especially when many nodes are processed. We would expect the node bound computation step of this algorithm to be considerably faster than solving an exact LP at a given node. Even if the optimal basis is passed to the exact LP solver as a warm start it would compute the final basic solution exactly, which may require solving a system of equations exactly. Computing a basic solution exactly is likely to be slower than using a precomputed LU factorization to make the correction in Algorithm 3.

3.2 Generality and Bound Quality

We now show that the assumption that the dual problem had no implied equalities and a full row rank constraint matrix is more general than the assumption that all primal variables have finite upper and lower bounds. For a problem described as equality constraints and non-negativity constraints on the variables the term *implied equality* refers to any variable bound that is implied to be tight.

Proposition 3.1. *The dual LP of a primal problem with finite lower and upper bounds l, u on its variables can be written in the form:*

$$\begin{aligned} \min \quad & b^T y - l^T z_l + u^T z_u \\ s.t. \quad & A^T y \ - I z_l \ + I z_u = c \\ & y, z_l, z_u \geq 0 \end{aligned}$$

If dual LP is feasible, then it has no implied equalities and the constraint matrix has full row rank.

Proof. The constraint matrix has full row rank because it has an identity submatrix. We now show that there are no implied equalities. Let (y, z_l, z_u) be a feasible solution and let $\alpha = \sum_{i=1}^{m} (A^T)_i$ (the sum of all the columns of A^T). Consider the solution $(y + \mathbb{1}, z_l + \mathbb{1} + \alpha^+, z_u + \mathbb{1} + \alpha^-)$. We can see that each component is at least one and $A^T(y+\mathbb{1}) - I(z_l + \mathbb{1} + \alpha^+) + I(z_u + \mathbb{1} + \alpha^-) = A^T y + \alpha - I z_l - \alpha^+ + I z_u + \alpha^- = A^T y - I z_l \ + I z_u = c$. Therefore this gives a feasible solution that is strictly positive in each component verifying that no variables are implied to be zero. □

Moreover, as we will see in Section 4, when considering our test set of 59 MIP instances, the conditions that the dual problem, at the root node, has no implied equalities and that the dual constraint matrix has full rank holds on 57 of them, where as only 28 of the instances had bounds on all primal variables.

The project-and-shift method relies on the existence of a full row rank submatrix A_S^T and also on the existence of a corrector point y^* that is S-interior. We now show that the existence of the S-interior point is equivalent to the condition that there are no implied equalities.

Proposition 3.2. *Suppose the LP* $\min\{b^T y | A^T y = c,\ y \geq 0\}$ *is feasible and* A^T *has full row rank. Then there exists an* S*-interior point for a full row rank subset of columns* S *of* A^T *if and only if there are no implied equalities.*

Proof. First, suppose there is an S-interior point of a full rank subset of the columns S. We may assume that $A^T = [A_S^T | A_N^T]$ where N is the set of columns not in S and there is a solution (y_S, y_N) with $y_S > 0$. Let $i \in N$ and suppose $y_i = 0$, then we can construct a solution in the following way. Since A_S^T has full row rank there exists a solution w to the equations $A_S^T w = A_i^T$. We can choose $\epsilon > 0$ such that $(y_S - \epsilon w) > 0$ and then $(y_S - \epsilon w,\ y_N + \epsilon e_i)$, where e_i is the ith unit vector, is a feasible S-interior point that is strictly positive in component i. This can be repeated for any column in N and therefore there are no implied equalities. The converse of the statement holds trivially by taking S equal to all the columns. □

Now we consider the quality of the bounds that are produced by the project-and-shift algorithm. Proposition 3.3 gives a bound on the strength of the LP bound in terms of the approximate dual solution and the information computed in Algorithm 2.

Proposition 3.3. *Assume that when Algorithm 2 is applied to the root node problem it identifies an* S*-interior point* y^* *with objective value* z_R*, and that* $\forall i \in S$, $y_i^* \geq d > 0$*. Next, suppose Algorithm 3 is applied at the node to compute a bound, given an approximate solution* $\tilde{y}, \tilde{z} \geq 0$ *with objective value* $\tilde{z}_N = b'^T \tilde{y} + \bar{b}^T \tilde{z}$*. Also assume that* $(z_R - \tilde{z}_N) \geq 0$ *(otherwise* z_R *can be taken as a safe dual bound). Let* $r = c - A^T \tilde{y} - \bar{A}^T \tilde{z}$ *be the error of the approximate solution and let* w *be the correction used,* $A^T w = r$*. Then the bound returned will be at most*

$$\tilde{z}_N + (1/d)(\max_{i \in S} w_i^-)(z_R - \tilde{z}_N) + (b'^T w)^+.$$

Proof. First note that $(1/d)(\max_{i \in S} w_i^-)$ gives an upper bound on the value of λ computed in Algorithm 3. Now we overestimate the dual bound produced by Algorithm 3.

$$\begin{aligned} b'^T y + \bar{b}^T z &= (1-\lambda)(b'^T(\tilde{y} + w) + \bar{b}^T \tilde{z}) + \lambda(b'^T y^*) \\ &\leq (1-\lambda)\tilde{z}_N + (1-\lambda) b'^T w + \lambda z_R \\ &\leq \tilde{z}_N + \lambda(z_R - \tilde{z}_N) + (b'^T w)^+ \\ &\leq \tilde{z}_N + (1/d)(\max_{i \in S} w_i^-)(z_R - \tilde{z}_N) + (b'^T w)^+ \end{aligned}$$ □

Unlike Proposition 2.1 and the primal-bound-shift method, project-and-shift does not necessarily depend on the values or existence of primal variable bounds. From Proposition 3.3

we can identify conditions that will likely produce stronger bounds. If the S-interior point y^* has a good objective value, and if its components in S have large values, this can improve the bound quality. Another desirable feature is that the projection vector w should be of small absolute value; this may be harder to control as w will depend on the solution to the system of equations $A^T w = r$. If the approximate dual solution only violates the constraints by a small amount, this will generally lead to a smaller difference between the objective value of the approximate dual solution and the bound value.

3.3 Generating Projections

A key component of the project-and-shift method is the projection step. A projection of the approximate dual solution into the affine hull of the dual polyhedron ensures that all equality constraints are satisfied. Projection of a vector into an affine space is a standard operation of linear algebra. In this section we explain our strategy for computing projections.

As described in Algorithms 2 and 3, the projection is done by computing an LU factorization of a rectangular matrix A_S^T, which is used to compute a corrector w by solving $A^T w = r$. Using an LU factorization can take advantage of sparsity of the matrix, which is often very high in real world MIP instances. Computing LU factorizations on problems arising from LP applications is a well studied area so we can take advantage of these highly developed techniques. The matrix factorization, which is the most difficult operation, is only performed once during Algorithm 2 in the setup stage. When Algorithm 3 is called to compute node bounds, the projection is accomplished by performing a back-solve using the already computed factorization.

Alternative strategies for computing projections could use other methods, such as orthogonal projections. An advantage of computing orthogonal projections is that the approximate solution would be mapped to the closest point in the affine hull. The drawback is that they may be significantly more expensive to compute. To symbolically compute an orthogonal projection at each node may be even more difficult than calling the exact LP solver with warm starts.

In the project-and-shift method we have a degree of freedom in choosing a subset of dual columns S, determining which components are adjusted during the projection. Choosing S as all the columns is a valid choice, but there are reasons to select a smaller subset. Ideally we would pick the columns of S as those dual variables which can be adjusted without having a large effect on the objective value. One strategy is to consider the optimal primal solution

at the root node and then choose S to be the set of all dual columns corresponding to active primal constraints (constraints that are satisfied with equality by the solution). If we correctly compute the optimal primal solution at the root node this choice of S would give a full row rank submatrix A_S^T. We think that this selection leads to tighter dual bounds than taking all dual columns. The reason is that the impact of such variables on the dual objective function via their objective coefficients is probably smaller than the impact of dual variables that, for example, correspond to non-active (large) primal bound constraints. Adjusting such variables may change the dual objective value to a smaller degree. The selection strategy will be studied experimentally in Section 4.

3.4 Identifying an S-interior Point

Freund et al. (1985) described a method to simultaneously compute an interior point and identify the affine hull of a polyhedron by solving a single LP. Using a similar idea we write an auxiliary problem that will identify an S-interior point if one exists for a set S. Expressing the problem in the dual form, implied equality constraints correspond to components of the problem that must be zero in any feasible solution. Here the added variables are $\delta \in \mathbb{R}^{|S|}$ and $\lambda \in \mathbb{R}$.

Dual LP Problem:

$$\begin{aligned} \min \quad & b^T y \\ \text{s.t.} \quad & A^T y = c \\ & y \geq 0 \end{aligned}$$

Auxiliary LP Problem:

$$\begin{aligned} \max \quad & \sum_{i \in S} \delta_i \\ \text{s.t.} \quad & A^T y - \lambda c = 0 \\ & y_i \geq \delta_i \quad \forall i \in S \\ & y \geq 0, \lambda \geq 1, \ 0 \leq \delta_i \leq 1 \end{aligned}$$

If we set S equal to all the columns of A^T and suppose $P = \{y | A^T y = c, y \geq 0\}$ and (y, δ, λ) is an optimal solution to the auxiliary problem then $\frac{1}{\lambda} y$ is contained in the relative interior of P, and $\delta_i = 0$ implies $y_i = 0$ for every $y \in P$. Geometrically this adds an extra dimension λ which scales the right hand side of the constraints converting the polyhedron to a conic form. The variables δ are indicators of each inequality being satisfied strictly, and if any δ_i can take a nonzero value, it can attain its maximum value of 1 by increasing (y, λ) and moving further into the cone. Choosing S to be any strict subset of the dual columns, an optimal solution would produce an S-interior point $\frac{1}{\lambda} y$ if one exists; otherwise some variables δ_i for $i \in S$ would be zero.

In Algorithm 1 it was necessary to identify the affine hull and an interior point of the dual polyhedron. Solving this auxiliary problem exactly would accomplish these goals. However, it would not be practical to solve for each bound computation because that would require the solution of an exact LP at each node.

We can use this auxiliary LP problem within Algorithm 2 to correctly identify y^* as needed. It would also recognize if no S-interior point exists if δ_i was equal to zero for any $i \in S$ in the optimal solution. By Proposition 3.2 if this auxiliary problem fails to find an S-interior point then the problem has implied equalities.

The disadvantage of using this auxiliary problem to identify the S-interior point is that the point is chosen in an arbitrary way. It could have a very bad objective value and could also have some components with strictly positive but very tiny values that could result in poor bound values after application of Algorithm 3; both of these situations were observed on some problems in our test set. Next we consider ways of choosing an S-interior point while considering both its objective value and the value of its positive components.

Computing a bound using the project-and-shift method requires identification of an S-interior point. In Proposition 3.3 we see that the value of the bound computed will depend on the objective value of this point, and the magnitude of the shift will depend on how large the values of the point are. Therefore the ideal point y^* would have a good objective value and also have large values in the components in S. Since we have assumed that S induces a full row rank submatrix of A^T and there exists an S-interior point we can use the following modified auxiliary LP to identify one, where α is a weight given to balance the two components of the objective function. We just introduce one additional variable δ that is a lower bound on the entries of S. We also include an upper bound M on δ to avoid having an unbounded problem.

Optimized Auxiliary LP Problem:

$$\begin{aligned} \max \ & (1-\alpha)(\max\{1, |z_{LP}|\})\delta + \alpha(b^T y) \\ \text{s.t.} \ & A^T y = c \\ & y_i \geq \delta \quad \forall i \in S \\ & y \geq 0, \ 0 \leq \delta \leq M \end{aligned}$$

The objective function of the optimized auxiliary LP problem is a convex combination of two objectives. First, $(1-\alpha)(\max\{1, |z_{LP}|\})\delta$, corresponds to maximizing a lower bound δ on the minimum value over components in S. The second part $\alpha(b^T y)$ corresponds to

the objective value of the original dual problem. One option for solving this problem is to simply choose the weight α and solve the problem once using an exact LP solver. We would typically have access to at least an approximation of the optimal solution for the root node LP so we normalize the first term by including a factor of $\max\{1, |z_{LP}|\}$ where z_{LP} is the optimal objective value at the root node, or an approximation thereof.

In our experiments we found this strategy to be successful for some problems, but in other cases the optimal solution had a value of $\delta = 0$, even when setting the value of α to be very small. When $\delta = 0$ this indicates that, due to too much emphasis on the second part of the objective function, the solution does not give an S-interior point although there is one; this then leads to failure of the project-and-shift method. In Table 1 we list some of the failure rates for different values of α.

Table 1: Success rate of single stage optimized auxiliary LP.

Value of α	Failure Rate ($\delta = 0$)
$\alpha = 0.1$	17/59
$\alpha = 0.01$	8/59
$\alpha = 0.001$	7/59
$\alpha = 0.0001$	2/59

After observing this behavior we employed a second strategy where the problem is solved in two stages. First, the problem was solved by setting $\alpha = 0$. Then knowing a feasible value δ^* of δ, the lower bound M is set equal to δ^*. Second, with this lower bound on δ, α is set equal to 1 and the problem is re-solved. Solving the second problem can also be done as a reoptimization since the only modification to the problem changed the variable bounds and objective. After solving the first step of this problem and adjusting the lower bound on δ, the optimal basis is still primal feasible (but not dual feasible) and therefore re-optimization can be done using the primal simplex algorithm.

3.5 Shifting by Interior Ray

The shifting step of the project-and-shift method corrects, in Algorithm 3, a projected approximate solution by shifting toward a corrector point y^*. An alternative way to correct the projected solution would be to add a multiple of a ray from the dual recession cone to correct it. We could adjust Algorithm 2 so that instead of finding an S-interior point y^* it would compute a ray r^* in the dual recession cone, $R = \{r : A^T r = 0, r \geq 0\}$, satisfying $r_i^* > 0 \; \forall i \in S$.

Following our previous notation, we would call this an *S-interior ray*. Then, in Algorithm 3, instead of computing a bound from the corrected solution $(y, z) = (1-\lambda)(\tilde{y}+w, \tilde{z})+\lambda(y^*, 0)$ we would use $(y, z) = (\tilde{y} + w, \tilde{z}) + \gamma(r^*, 0)$, where γ is chosen large enough that $(y, z) \geq 0$.

The drawback of this strategy is that it is less general than the previous assumptions. For example, if the root node dual LP has a bounded feasible region, a ray r^* satisfying these conditions does not exist.

Proposition 3.4. *Existence of an S-interior ray in the dual is implied by presence of all primal bounds. Existence of an S-interior ray implies existence of an S-interior point.*

Proof. If the primal problem has upper and lower bounds on all variables then the dual can be expressed as $\min\{b^T y - l^T z_l + u^T z_u | A^T y \ - I z_l \ + I z_u = c, \quad y, z_l, z_u \geq 0\}$. Then the ray given by $(\mathbb{1}, \mathbb{1} + \alpha^+, \mathbb{1} + \alpha^-)$, where $\alpha = \sum_{i=1}^{m} (A^T)_i$, is in the recession cone and is strictly positive in each component. To see that the existence of an S-interior ray implies the existence of an S-interior point we observe that taking any feasible solution and adding some multiple of an S-interior ray would give an S-interior point. □

Examples can easily be constructed to demonstrate that none of the reverse implications hold. Table 2 lists how often each of these conditions holds at the root node for our test set.

Table 2: Occurrence of conditions.

Condition	Occurrence
All primal bounds present	28/59
Existence of S-interior ray	48/59
Existence of S-interior point	57/59

3.6 Proving LP Infeasibility

In a branch-and-bound tree it is often necessary to certify primal infeasibility of the node LP relaxations. Primal infeasibility can be certified by proving that the dual problem is unbounded. Proving dual infeasibility / primal unboundedness is not as relevant a problem because if the root node primal LP has a bounded objective value, then it will remain bounded through the branch-and-bound tree. In this section we describe two different approaches of how the project-and-shift algorithm can be adapted to certify primal infeasibility.

One possible method for proving the dual objective value is unbounded is to have the floating-point LP solver return a cost-improving dual ray and then correct this approximate

ray to be exactly feasible. The dual ray could be projected and shifted to be exactly feasible and if the resulting exactly feasible dual ray was cost improving then primal infeasibility would be certified. The projection could be done similarly to Algorithms 2 and 3, except that the ray would be corrected to satisfy $A^T y = 0$, and the shift could be accomplished using an S-interior ray of the dual. This approach has some drawbacks. It would require an extra auxiliary problem to be solved exactly at the root node to identify an S-interior ray. It would also require the less general condition that there is an S-interior ray which was discussed in Section 3.5.

An alternate approach is to compute a valid dual bound strong enough to prune the node without explicitly finding a cost improving dual ray. Whenever a valid dual bound for a node is identified that surpasses the primal bound (the best known primal objective value), then that node can be pruned. In fact, many software packages for branch-and-bound will terminate the simplex algorithm early at a node after a dual solution good enough to prune the node is identified.

Therefore at a node which is claimed to be dual unbounded by the inexact LP solver, an approximate dual solution that surpasses the best primal bound can be identified and then the project-and-shift algorithm can be applied to that solution with the goal of pruning the node. An approximate dual solution surpassing the primal bound value should be readily available at a primal infeasible node. Such a dual solution could be returned directly by the floating-point LP solver, or it could be constructed by adding a multiple of an approximate cost improving dual ray to a dual feasible solution.

Depending on the quality of the approximate dual ray returned by the floating-point LP solver it may or may not be possible to certify dual infeasibility at a given node. Additionally, this method will not apply if a node is both primal and dual infeasible. In these cases we would resort to using the exact LP solver.

4. Computational Study

In this section we describe an implementation of the project-and-shift method and the resulting computational results. First we compare the behavior of several variations of this algorithm in practice. Secondly, we demonstrate that this method provides an advantage over other dual bounding methods on some classes of problems.

4.1 Implementation and Test Set

The project-and-shift method for generating valid LP bounds is implemented within a hybrid rational branch-and-bound version of the MIP software package SCIP (Achterberg, 2007, 2009). This hybrid exact MIP solver is described in (Cook et al., 2011) and uses a combination of exact rational arithmetic and safe floating-point computation to compute exact solutions. An exact representation of the problem is stored, but whenever possible computations are preformed on a floating-point relaxation or approximation of the original problem. The implementation can choose between multiple different methods to compute valid dual bounds, which must be computed for any binding decisions. This initial version of the rational MIP solver is a pure branch-and-bound implementation that uses the first fractional variable branching rule and the best bound node selection strategy.

The code is based on SCIP version 1.2.0.8. QSopt_ex 2.5.5 (Applegate et al., 2007b) is used as the exact LP solver and Cplex 12.2 (IBM ILOG, 2011) is used as the floating-point LP solver. The auxiliary LP in the setup phase of the project-and-shift algorithm is solved using the QSopt_ex interface. The rectangular LU factorizations used for projections are computed using a code developed by combining the exact LU factorization code of QSopt_ex (Applegate et al., 2007a) and a sparse numerical rectangular LU factorization code from (Dash and Goycoolea, 2010) which was provided by Sanjeeb Dash. Both of these codes were based on the methods described by Suhl and Suhl (1990).

All computations were performed on one of several identical Linux machines with Intel E5420 4-core processors and 16 GB of RAM. To maintain accurate timings, each computer was limited to running one test at a time. Computations are performed on a test set of 59 MIP problems selected from Miplib 3.0 (Bixby et al., 1998), Miplib 2003 (Achterberg et al., 2006), and the collection of Mittelmann (Mittelmann, 2006). The test set was selected by taking all instances which could be solved by the floating-point version of SCIP within two hours using pure branch-and-bound with the settings: first fractional branching, best bound node selection. A full listing of the problems in our test set appears in Table 5. When studying the performance of the project-and-shift method on these test problems, the time limit was increased to 24 hours.

One additional practical step that is taken in Algorithm 3 is a simple postprocessing of the exactly feasible dual solution obtained by the project-and-shift method when some constraints of the problem have right and left hand sides. If the dual multipliers for both

sides of a specific constraint are nonzero then they can be lowered by equal amounts so that one becomes zero, this will reduce the cost of the constructed dual solution and improve the bound quality.

4.2 Computations

We have described several variants of the project-and-shift algorithm, the significant decisions to be made are how to choose the set S and how to choose the S-interior point. We want a method that is as general as possible and is fast for the MIP application. Two things that can influence the overall speed of the MIP solver are; how fast the bounds can be computed and how strong the bounds are, because weak bounds may lead to an increased node count.

As a first consideration we eliminate some of the proposed variants due to their lack of generality. The use of an S-interior ray instead of an S-interior point described in Section 3.5 has conditions that were satisfied less often than the other versions.

Next we consider the optimized S-interior point described in Section 3.4. We described an auxiliary problem with a two part objective function. When using nonzero values of α we often experienced problems where the solution to the auxiliary problem was not S-interior, the failure rates are listed in Table 1. Based on these failure rates we will not consider these variants with $\alpha > 0$ as viable alternatives, however we do consider the setting of $\alpha = 0$ and the two stage problem. We also remark that in the cases where using nonzero values of alpha did work, the performance on the overall branch-and-bound tree was similar to the performance of $\alpha = 0$.

After eliminating these possibilities we are left with three choices of how to compute the S-interior point for a given set S. First, we could choose an arbitrary one using the auxiliary problem listed in the beginning of Section 3.4. Second, we could solve the optimized interior point problem given in Section 3.4 with $\alpha = 0$; maximizing the minimum value of components in S. Third, we could solve the two stage problem, where we first maximize the minimum over components in S, and then do further optimization with a modified objective function to improve the point's objective value. In the tables, we will denote these three settings as "P:Arb", "P:Opt" and "P:2Stage", respectively.

The second parameter we have to choose is how to select the set S. We consider two possibilities, first we can let S be equal to all the dual columns. The second possibility is to set S equal to all dual columns corresponding to primal constraints that are active at the optimal root node solution (determined by either the exact or approximate root node

Table 3: Relative bound quality and extra computation time (geom. mean) at root.

Setting	Zero	S	M	L	∞	DB Time (s)
S:Act;P:Opt	19	32	6	0	2	2.7
S:Act;P:Arb	19	34	4	0	2	3.0
S:Act;P:2Stage	18	37	2	0	2	4.1
S:All;P:Opt	12	38	5	2	2	3.6
S:All;P:Arb	12	38	5	2	2	3.0
S:All;P:2Stage	12	40	3	2	2	5.9

LP). The motivation for such a choice is discussed in Section 3.3. We denote the parameter choices for the set S by "S:All" and "S:Act". These settings give us six combinations to compare.

4.3 Root Node Performance

First we compare the behavior at the root node, evaluating the quality of the bound produced and the time necessary to compute it. Note that the dual bounding time required at the root node is dominated by the solution of the auxiliary problem in Algorithm 2, which in each case involves solving one or two exact LPs. Table 3 compares the relative quality of the dual bounds at the root node. The bound quality is measured as the relative difference $d = (\overline{c_{LP}} - \overline{c_b})/\max\{1, \overline{c_{LP}}, \overline{c_b}\}$ where c_{LP} is the exact optimal value of the root node LP, c_b is the bound value, and the overline notation represents the floating-point upper approximation of these numbers. The FP upper approximations are used because this is how the values are compared in the numeric component of the hybrid symbolic-numeric implementation in (Cook et al., 2011). For each setting we list how many problems had bound quality in different ranges where "Zero" is no difference, "S" indicates $d \in (0, 10^{-9}]$, "M" indicates $d \in (10^{-9}, 10^{-3}]$, "L" indicates $d \in (10^{-3}, \infty)$, and "∞" indicates that no finite bound was returned. The column marked "DB Time" in Table 3 gives the geometric mean of the time required to compute the dual bound. Bound computations requiring less than one second are rounded up to one second, this occurs on more than half of the problems.

Figure 1 represents the dual bound computation time in a performance profile. A performance profile is a representation of the solution times which plots the distribution of the ratio of solution times for each solver when compared to whichever solver was the fastest for each instance. The x-axis represents how many times slower the solvers were, and the

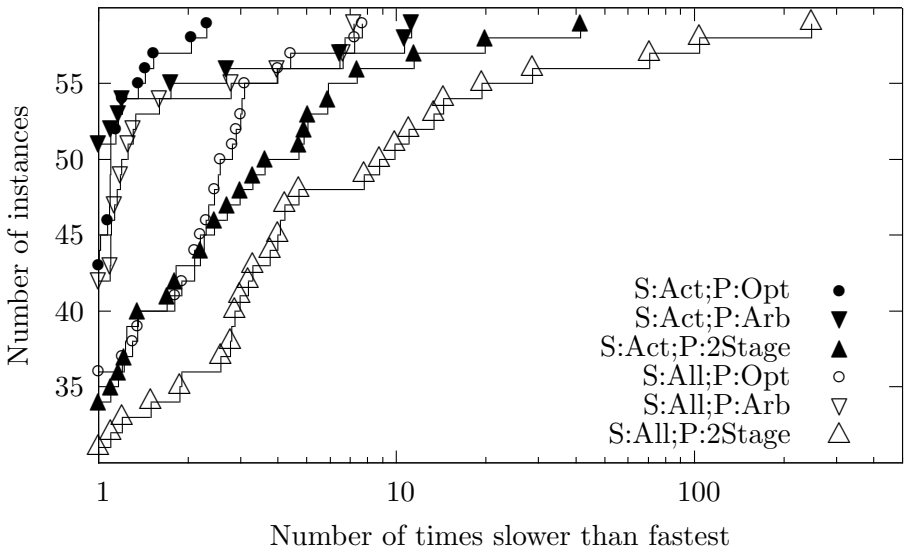

Figure 1: Comparison of extra time for safe bounds at root node.

y-axis depicts the number of instances. Dolan and Moré (2001) given a more detailed description of performance profiles and discuss their strengths for visually representing results of optimization solvers.

We now remark on the relative quality of the bounds produced by these settings. Choosing S equal to the active columns leads to increased bound quality, as was predicted. The selection of the interior point also impacts the quality, with the arbitrary interior point giving the worst bounds, and the two stage problem producing better bounds on average. Regarding the root node computation time, the two stage problem leads to a significant increase in the solution times compared to the other methods. The other selections of the S-interior point lead to varied computation time; the setting "P:Opt" could be faster in some cases due because its auxiliary problem has less variables, and the setting "P:Arb" may be faster in others because it may require less pivots to solve the auxiliary LP problem. Also, in general, choosing S equal to the active columns instead of all columns reduces the solution time.

4.4 Branch-and-Bound Performance

Table 4 gives a summary of the overall performance in the branch-and-bound process of the six variations of the project-and-shift method, compared to the dual bounding strategy of solving each node LP exactly, denoted "ExactLP". The table includes how many of the problem instances were solved within the 24h time limit. Then for the 49 problems solved

Table 4: Summary of overall performance.

		Geometric mean for instances solved by all settings (49)		
Setting	slv	Nodes	Time (s)	DB Time (s)
S:Act;P:Opt	50	19 345	453.1	308.8
S:Act;P:Arb	50	19 335	471.8	324.8
S:Act;P:2Stage	50	19 329	502.4	376.5
S:All;P:Opt	50	19 452	454.9	307.5
S:All;P:Arb	50	20 194	475.4	318.3
S:All;P:2Stage	50	19 562	542.3	417.1
ExactLP	52	19 169	846.2	679.3

by each of these methods we display the geometric means of the node counts, solution time, and the amount of solution time used for computing safe dual bounds.

Further details are given in Table 5, which shows the individual solution times and Table 6, which gives the number of nodes required to solve each MIP. Solution times that are within 5% of the fastest method have been made bold in Table 5. Instances containing variables with upper bound $u = \infty$ or lower bound $l = -\infty$ are indicated by a "×". Finally, Figure 2 gives a performance profile comparing the individual solution times. In order to make this figure more readable, not all of the settings are included.

Concerning the overall performance we first observe that "ExactLP" is slower than the project-and-shift variants by a factor of nearly 2 in geometric mean. When comparing the variants of the project-and-shift algorithm we observe that the setting "S:Act;P:Opt" has the fastest average solution time, but is closely followed by "S:All;P:Opt". One explanation of why choosing all columns may lead to faster solution times in some cases is as follows. If we restrict the column choice, this will change the behavior of the LU factorization code, possibly resulting in more fill-in and longer solution times. In contrast, if the rectangular LU factorization code is working with all columns of the dual it could pivot on the sparsest columns, i.e., those coming from the existing primal variable bounds.

On problems solved to optimality, the node counts were often similar between the different methods. This indicates that although the project-and-shift method is producing LP bounds that are not as tight as the exact LP solutions, they are often good enough that they do not significantly increase in the number of branch-and-bound nodes. However, there were some specific instances where the node counts differed considerably between methods such as

Table 5: Time (s) needed to solve each problem instance.

Example	S:Act;P:Opt	S:Act;P:Arb	S:Act;P:2Stage	S:All;P:Opt	S:All;P:Arb	S:All;P:2Stage	ExactLP
30:70:4_5:0_95:100	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	**76.9**
acc-0	3.3	**3.1**	6.0	4.6	3.3	6.7	4.1
acc-1	**337.1**	**335.8**	**342.0**	**338.6**	**338.3**	**346.8**	433.0
acc-2	**45.8**	**45.7**	50.1	**47.8**	**45.7**	54.8	56.5
air03	30.1	24.4	31.6	54.7	26.6	62.7	**3.4**
air05	15474.3	15114.2	16131.9	**14036.7**	**13883.2**	14638.0	19348.7
× bc1	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0
× bell3a	564.4	**527.4**	**542.7**	684.0	704.2	680.3	3674.9
× bell5	480.0	**447.9**	473.6	578.1	595.1	584.2	3671.0
× bienst1	**296.0**	328.6	**309.8**	**301.9**	**299.5**	**305.9**	811.1
× bienst2	**3768.6**	4271.1	3943.7	**3698.5**	**3689.5**	**3713.9**	8613.2
× blend2	**207.0**	241.8	226.8	253.8	248.8	239.8	583.5
× dano3_3	121.2	394.0	424.6	**95.9**	250.4	612.0	401.8
× dano3_4	415.8	749.5	705.0	**246.3**	484.8	762.0	1864.3
× dano3_5	8033.9	8496.8	8908.5	**3680.9**	3967.0	4446.2	62674.6
× dcmulti	**115.6**	**116.9**	**116.2**	**114.4**	**113.1**	**114.8**	215.3
× dsbmip	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0
× egout	**121.5**	**119.5**	**125.4**	129.2	130.4	131.6	359.6
eilD76	14369.9	14236.4	14383.4	12029.5	**11397.3**	12300.7	16786.7
enigma	162.7	153.7	**143.8**	**146.4**	**141.4**	**146.2**	353.1
× flugpl	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	1.6
× gen	**397.0**	**401.8**	**384.8**	421.9	440.6	424.4	613.4
× gesa3	**2645.3**	**2571.5**	2729.2	2749.5	2793.2	2756.3	4696.8
× gesa3_o	3366.9	3299.2	3487.2	**2817.7**	**2825.2**	**2893.5**	5860.4
irp	**33107.1**	**31734.0**	**33192.6**	41963.4	38911.5	41982.9	43510.8
× khb05250	**27.7**	**27.9**	**27.9**	29.2	**27.7**	**27.9**	43.5
l152lav	332.3	332.9	335.1	341.5	344.8	342.3	**279.1**
lseu	**730.1**	**695.5**	737.8	870.2	868.0	860.1	865.6
× markshare1_1	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0
× markshare4_0	1551.6	1599.1	1571.0	**1274.6**	**1246.2**	**1270.7**	22074.2
mas76	82284.8	>86400.0	73859.6	**41521.3**	51570.2	**41074.9**	>86400.0
mas284	72074.8	76158.6	66608.3	**19623.2**	70957.7	**20041.7**	47618.4
× misc03	**4.0**	**4.0**	**3.9**	4.9	4.7	4.9	4.7
× misc06	**14.2**	**13.7**	**14.2**	14.6	**14.1**	15.1	283.8
× misc07	**2550.8**	**2500.0**	**2534.3**	2769.0	2769.5	2815.7	3564.5
mod008	283.7	**246.0**	285.2	**251.3**	**246.3**	**251.6**	587.5
mod010	2911.3	2909.7	2858.7	3409.6	3409.1	3440.1	**2682.4**
× mod011	67906.5	69352.2	70488.6	78352.3	**61957.5**	69642.6	80443.6
neos5	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	**63644.7**
neos8	**8593.2**	**8691.4**	**8695.3**	9602.5	9222.7	12841.2	72125.8
× neos11	**2714.3**	**2778.9**	**2671.2**	2823.3	**2729.9**	**2744.9**	3059.2
× neos21	**15662.0**	**15680.2**	**16204.2**	16833.9	**16168.0**	18013.6	23479.2
neos897005	706.1	577.5	933.8	1188.0	570.2	13551.0	**392.8**
nug08	**19.0**	**18.5**	38.0	20.5	**19.3**	47.1	42.3
nw04	16235.5	16321.4	16609.8	19473.7	17320.9	20553.9	**12097.7**
p0033	**1.0**	**1.0**	**1.0**	1.3	1.3	1.3	1.3
p0201	19.5	**17.8**	**17.8**	19.8	19.3	19.8	30.5
× pk1	6509.8	7461.9	6490.5	**3665.1**	**3614.5**	**3602.8**	21516.5
qap10	**573.9**	**572.0**	697.1	**575.9**	**569.1**	1149.5	2120.3
× qnet1_o	**12195.0**	12824.0	**12177.0**	13677.7	14478.3	13598.0	19706.4
× ran13x13	>86400.0	**79314.8**	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0
× rentacar	70.3	156.5	254.0	66.4	436.4	109.2	**47.3**
rgn	28.9	**26.9**	28.7	33.0	33.2	33.7	145.2
stein27	**2.8**	**2.7**	**2.8**	3.1	3.1	3.0	4.9
stein45	**97.4**	**96.6**	**97.5**	112.3	108.0	108.7	182.0
× swath1	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	**69978.7**
× swath2	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0
vpm1	25645.6	23236.9	24279.5	27183.0	24578.9	27156.8	**20405.2**
vpm2	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0	>86400.0

Table 6: Branch-and-bound nodes needed to solve each problem instance.

Example	S:Act;P:Opt	S:Act;P:Arb	S:Act;P:2Stage	S:All;P:Opt	S:All;P:Arb	S:All;P:2Stage	ExactLP
30:70:4_5:0_95:100	>250 119	>305 404	>252 550	>276 163	>329 978	>279 885	190
acc-0	52	52	52	52	52	52	52
acc-1	3 224	3 224	3 224	3 224	3 224	3 224	3 224
acc-2	241	241	241	241	241	241	241
air03	21	21	21	21	21	21	21
air05	94 269	94 269	94 274	94 269	94 269	94 274	94 283
× bc1	>225 758	>171 680	>214 798	>67 577	>85 302	>71 891	>52 189
× bell3a	362 609	362 608	362 611	362 620	362 618	362 621	362 615
× bell5	408 929	409 007	409 000	408 938	408 960	409 000	408 992
× bienst1	42 018	42 017	42 017	41 892	41 889	41 895	40 898
× bienst2	447 178	447 177	447 176	447 285	447 282	447 283	447 177
× blend2	44 988	44 990	44 988	45 006	44 986	44 989	44 992
× dano3_3	40	40	40	40	40	40	40
× dano3_4	193	193	193	193	193	193	193
× dano3_5	4 722	4 718	4 720	4 726	4 726	4 724	4 718
× dcmulti	20 133	20 133	20 133	20 133	20 133	20 133	20 133
× dsbmip	>695 513	>689 797	>679 733	>674 224	>682 417	>679 723	>191 210
× egout	60 871	60 871	60 871	60 871	60 871	60 871	60 871
eilD76	236 305	236 305	236 305	236 305	236 305	236 305	236 303
enigma	128 058	128 058	128 058	128 058	128 058	128 058	128 058
× flugpl	3 519	3 519	3 519	3 519	3 519	3 519	3 519
× gen	34 100	34 100	34 100	34 100	34 100	34 100	34 100
× gesa3	128 210	128 210	128 210	128 210	128 210	128 210	128 210
× gesa3_o	178 437	178 437	178 437	178 437	178 437	178 437	178 437
irp	116 177	116 182	116 177	116 177	116 163	116 177	116 177
× khb05250	6 606	6 606	6 606	6 606	6 606	6 606	6 606
l152lav	11 933	11 929	11 933	11 933	11 930	11 933	11 934
lseu	795 963	795 963	795 963	795 955	795 944	795 961	795 963
× markshare1_1	>126 148 808	>126 736 454	>126 601 178	>156 379 267	>156 943 548	>156 918 719	>10 278 529
× markshare4_0	3 826 128	3 826 128	3 826 128	3 826 128	3 826 128	3 826 128	3 826 096
mas76	7 568 599	>7 357 073	7 265 136	10 425 959	7 415 304	10 312 915	>3 593 826
mas284	1 894 754	1 709 650	1 801 863	2 493 189	7 556 747	2 475 923	1 709 652
× misc03	1 561	1 561	1 561	1 561	1 561	1 561	1 559
× misc06	306	306	306	306	306	306	255
× misc07	368 179	368 179	368 180	368 182	368 181	368 175	367 676
mod008	59 211	59 211	59 211	59 211	59 211	59 211	59 211
mod010	93 732	93 748	93 730	93 731	93 741	93 730	93 730
× mod011	421 651	421 651	421 651	421 651	421 650	421 651	421 651
neos5	>28 412 949	>29 163 913	>28 757 034	>40 773 418	>41 780 866	>40 946 550	26 371 494
neos8	24 928	24 930	24 936	24 928	24 928	24 928	25 091
× neos11	32 006	32 006	32 006	32 004	32 004	32 004	30 020
× neos21	830 716	830 716	830 716	830 726	830 718	830 726	818 611
neos897005	86	86	86	86	86	86	86
nug08	143	143	143	143	143	143	143
nw04	10 826	10 815	10 826	10 826	10 818	10 826	10 826
p0033	2 670	2 670	2 670	2 670	2 670	2 670	2 670
p0201	5 788	5 780	5 780	5 780	5 780	5 780	5 780
× pk1	1 793 664	1 793 664	1 793 664	1 793 664	1 793 664	1 793 664	1 793 656
qap10	246	246	246	246	246	246	246
× qnet1_o	730 464	730 655	730 465	730 424	730 378	730 428	731 031
× ran13x13	>26 422 361	27 604 880	>26 452 160	>25 411 352	>26 326 915	>25 422 590	>27 372 116
× rentacar	165	179	167	165	341	219	156
rgn	10 249	10 249	10 249	10 249	10 249	10 249	10 219
stein27	4 031	4 031	4 031	4 031	4 031	4 031	4 031
stein45	58 329	58 329	58 329	58 333	58 331	58 333	58 333
× swath1	>1 677 398	>1 665 414	>1 647 016	>1 685 890	>1 586 738	>1 684 253	560 996
× swath2	>1 664 965	>1 681 828	>1 664 681	>1 659 503	>1 658 897	>1 639 540	>716 964
vpm1	7 773 158	7 773 158	7 773 158	7 773 158	7 773 158	7 773 158	7 773 158
vpm2	>21 823 235	>23 283 669	>23 342 584	>22 477 542	>23 009 667	>21 671 938	>17 032 855

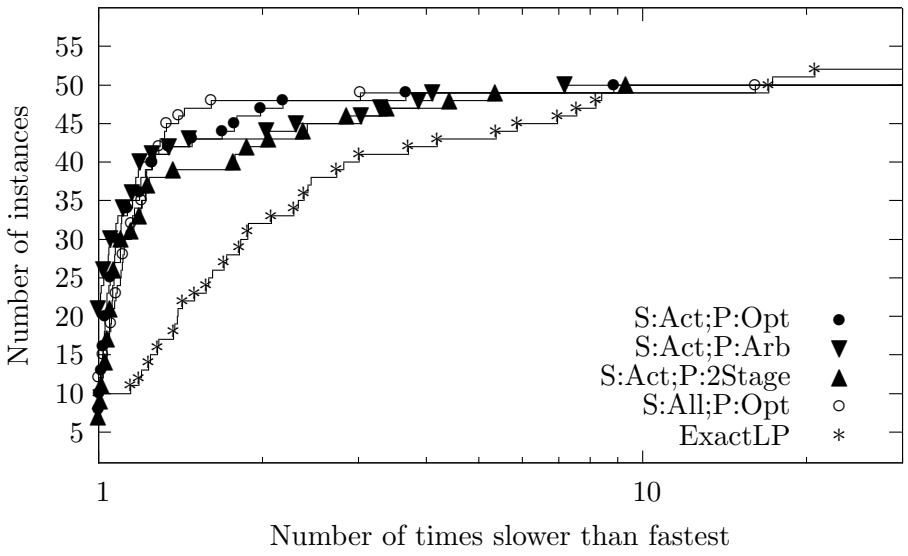

Figure 2: Comparison of time needed to solve each problem instance.

`rentacar`. We also note that for some instances the "ExactLP" method uses a small number of extra nodes when compared to some versions of the project-and-shift method. This may seem counterintuitive because the exact LP solver will produce the tightest possible bound at each node, but this phenomenon is explained by the use of the best-bound node selection rule where differing bounds will change the order in which nodes are processed, possibly leading to a small variation in node counts. We can also note that in Table 4, "ExactLP" processes fewer nodes on average than any of the other methods, as would be expected.

5. Conclusion

We have described a new method for computing valid dual bounds. The dual bounds are computed without requiring the primal LP to have upper and lower bounds on all variables. It needs the exact solution to an LP at the root node and an exact LU factorization, but once this information is computed the LP bounds at each node of the branch-and-bound tree can be computed quickly. We demonstrated this method to be more generally applicable than the primal-bound-shift method, and faster than solving an exact LP at each node. We also remark that the project-and-shift method is capable of computing bounds in a branch-and-cut framework, although this has not yet been tested computationally.

The fact that the conditions required by the project-and-shift method were satisfied on most instances also says something about the problem structure of the MIPs contained in

our test set. Namely, the conditions that the dual constraint matrix has full row rank and that none of the dual inequalities are implied equalities are often satisfied on these real-world problems.

There are possible future directions that could be explored to improve the speed of the methods developed in this article, or to apply similar ideas in other places. One possible future direction would be looking at a combination of the project-and-shift method and the related method of Althaus and Dumitriu (2009). First, the auxiliary LP to identify the polyhedral structure given in Section 3.4 could be used in place of the iterative algorithm used by these authors to determine implied equalities of the system. It is also possible that interval methods could be applied to the project-and-shift method to eliminate some of the exact computation and further increase the speed.

As a final remark we note that the paper Cook et al. (2011) contains a detailed computational study comparing the project-and-shift method with several other dual bounding methods. It also describes an sophisticated automatic strategy for switching between these different dual bounding methods at each node of the branch-and-bound tree. The project-and-shift method is an important component of this selection strategy, further validating its usefulness.

References

Achterberg, T. 2007. Constraint integer programming. Ph.D. thesis, Technische Universität Berlin.

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation* **1** 1–41.

Achterberg, T., T. Koch, A. Martin. 2006. MIPLIB 2003. *Operations Research Letters* **34** 361–372.

Althaus, E., D. Dumitriu. 2009. Fast and accurate bounds on linear programs. *Proceedings of 8th International Symposium on Experimental Algorithms (SEA 2009)* **5526** 40–50.

Applegate, D. L., R. E. Bixby, V. Chvátal, W. J. Cook. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, New Jersey, USA.

Applegate, D. L., W. Cook, S. Dash, D. G. Espinoza. 2007a. Exact solutions to linear programming problems. *Operations Research Letters* **35** 693–699.

Applegate, D. L., W. Cook, S. Dash, D. G. Espinoza. 2007b. QSopt_ex. `http://www.dii.uchile.cl/~daespino/ESolver_doc/main.html`.

Bixby, R. E., S. Ceria, C. M. McZeal, M. W. P. Savelsbergh. 1998. An updated mixed integer programming library: MIPLIB 3.0. *Optima* **58** 12–15.

Cook, W., T. Koch, D. Steffy, K. Wolter. 2011. An exact rational mixed-integer programming solver. *To appear in IPCO 2011: Integer Programming and Combinatorial Optimization* .

Dash, S., M. Goycoolea. 2010. A heuristic to generate rank-1 GMI cuts. *Mathematical Programming Computation* **2** 231–257.

Dhiflaoui, M., S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schömer, R. Schulte, D. Weber. 2003. Certifying and repairing solutions to large LPs: How good are LP-solvers? *SODA '03: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, PA, USA, 255–256.

Dolan, E. D., J. J. Moré. 2001. Benchmarking optimization software with performance profiles. *Mathematical Programming* **91** 201 – 213.

Espinoza, D. G. 2006. On linear programming, integer programming and cutting planes. Ph.D. thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology.

Freund, R. M., R. Roundy, M. J. Todd. 1985. Identifying the set of always-active constraints in a system of linear inequalities by a single linear program. *Technical Report, Sloan School of Management, MIT* .

IBM ILOG. 2011. CPLEX. `http://www.ilog.com/products/cplex`.

Jansson, C. 2004. Rigorous lower and upper bounds in linear programming. *SIAM Journal on Optimization* **14** 914–935.

Keil, C., C. Jansson. 2006. Computational experience with rigorous error bounds for the netlib linear programming library. *Reliable Computing* **12** 303–321.

Koch, T. 2004. The final NETLIB-LP results. *Operations Research Letters* **32** 138–142.

Kwappik, C. 1998. Exact Linear Programming. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken.

Mittelmann, H. D. 2006. LPtestset. `http://plato.asu.edu/ftp/lptestset/`.

Neumaier, A., O. Shcherbina. 2004. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming* **99** 283–296.

Steffy, D. E. 2011. Topics in exact precision mathematical programming. Ph.D. thesis, Algorithms, Combinatorics and Optimization, Georgia Institute of Technology.

Suhl, U. H., L. M. Suhl. 1990. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing* **2** 325–335.