An Exact Rational Mixed-Integer Programming Solver

William Cook*[1], Thorsten Koch[2], Daniel E. Steffy*[1], and Kati Wolter**[2]

[1] School of Industrial and Systems Engineering,
Georgia Institute of Technology, Atlanta, GA,
bico@isye.gatech.edu,desteffy@gatech.edu
[2] Zuse Institute Berlin, Germany, {koch,wolter}@zib.de

Abstract We present an exact rational solver for mixed-integer linear programming that avoids the numerical inaccuracies inherent in the floating-point computations used by existing software. This allows the solver to be used for establishing theoretical results and in applications where correct solutions are critical due to legal and financial consequences. Our solver is a hybrid symbolic/numeric implementation of LP-based branch-and-bound, using numerically-safe methods for all binding computations in the search tree. Computing provably accurate solutions by dynamically choosing the fastest of several safe dual bounding methods depending on the structure of the instance, our exact solver is only moderately slower than an inexact floating-point branch-and-bound solver. The software is incorporated into the SCIP optimization framework, using the exact LP solver QSopt_ex and the GMP arithmetic library. Computational results are presented for a suite of test instances taken from the Miplib and Mittelmann collections.

1 Introduction

Mixed-integer programming (MIP) is a powerful and flexible tool for modeling and solving decision problems. Software based on these ideas is utilized in many application areas. Despite their widespread use, few available software packages provide any guarantee of correct answers or certification of results. Possible inaccuracy is caused by the use of floating-point (FP) numbers [14]. FP calculations necessitate the use of built-in tolerances for testing feasibility and optimality, and can lead to calculation errors in the solution of linear-programming (LP) relaxations and in the methods used for creating cutting planes to improve these relaxations.

Due to a number of reasons, for many industrial MIP applications near optimal solutions are sufficient. Cplex, for example, defaults to a relative MIP optimality tolerance of 0.001. Moreover, when data describing a problem arises from imprecise sources, exact feasibility is usually not necessary. Nonetheless,

* Research supported by NSF Grant CMMI-0726370, ONR Grant N00014-08-1-1104.
** Research funded by DFG Priority Program 1307 "Algorithm Engineering".

accuracy is important in many settings. Direct examples arise in the use of MIP models to establish fundamental theoretical results and in subroutines for the construction of provably accurate cutting planes. Furthermore, industrial customers of MIP software request modules for exact solutions in critical applications. Such settings include the following.

- Feasibility problems, e.g., chip verification in the VLSI design process [1].
- Compiler optimization, including instruction scheduling [22].
- Combinatorial auctions [21], where serious legal and financial consequences can result from incorrect solutions.

Optimization software relying exclusively on exact rational arithmetic has been observed to be prohibitively slow, motivating the development of more sophisticated techniques to compute exact solutions. Significant progress has been made recently toward computationally solving LP models exactly over the rational numbers using hybrid symbolic/numeric methods [7,10,12,16,17], including the release of the software package QSopt_ex [6]. Exact MIP has seen less computational progress than exact LP, but significant first steps have been taken. An article by Neumaier and Shcherbina [19] describes methods for safe MIP computation, including strategies for generating safe LP bounds, infeasibility certificates, and cutting planes. The methods they describe involve directed rounding and interval arithmetic with FP numbers to avoid incorrect results.

The focus of this article is to introduce a hybrid branch-and-bound approach for exactly solving MIPs over the rational numbers. Section 2 describes how rational and safe FP computation can be coupled together, providing a fast and general framework for exact computation. Section 3 describes several methods for computing valid LP bounds, which is a critical component of the hybrid approach. Section 4 describes an exact branch-and-bound implementation within SCIP [1,2] and includes detailed computational results on a range of test libraries comparing different dual bounding strategies. The exact solver is compared with an inexact branch-and-bound solver and observed to be only moderately slower.

2 Hybrid Rational/Safe Floating-Point Approach

Two ideas for exact MIP proposed in the literature, and tested to some extent, are the *pure rational approach* [7] and the *safe-FP approach* [9,19]. Both utilize LP-based branch-and-bound. The difference lies in how they ensure the computed results are correct.

In the pure rational approach, correctness is achieved by storing the input data as rational numbers, by performing all arithmetic operations over the rationals, and by applying an exact LP solver [12] in the dual bounding step. This approach is especially interesting because it can handle a broad class of problems: MIP instances described by rational data. However, replacing all FP operations by rational computation will increase running times noticeably. For example, while the exact LP solver QSopt_ex avoids many unnecessary rational computations and is efficient on average, Applegate et al. [7] observed a greater

slowdown when testing an exact MIP solver that relied on rational arithmetic and called QSopt_ex for each node LP computation.

In order to limit the degradation in running time, the idea of the safe-FP approach is to continue to use FP-numbers as much as possible, particularly within the LP solver. However, extra work is necessary to ensure correct decisions in the branch-and-bound algorithm. Correctness of certain computations can be ensured by controlling the rounding mode for FP operations. Valid dual bounds can often be obtained by post-processing approximate LP solutions; this type of safe dual bounding technique has been successfully implemented in Concorde [5] for the traveling salesman problem. A generalization of the method for MIPs is described in [19]. Furthermore, the idea of manipulating the rounding mode can be applied to cutting-plane separation. In [9], this idea was used to generate numerically safe Gomory mixed-integer cuts. Nevertheless, whether the safe-FP approach leads to acceptable running times for general MIPs has not been investigated. Although the safe-FP version of branch-and-bound has great advantages in speed over the pure rational approach, it has several disadvantages. Everything, including input data and primal solutions, is stored as FP numbers. Therefore, correct results can only be ensured for MIP instances that are given by FP-representable data and that have a FP-representable optimal solution if they are feasible. Some rationally defined problems can be scaled to have FP-representable data. However, this is not always possible due to the limited representation of floating-point numbers, and the resulting large coefficients can lead to numerical difficulties. The applicability is even further limited as the safe dual bounding method discussed in [19] requires, in general, lower and upper bounds on all variables. Weakness in the safely generated bound values may also increase the number of nodes processed by the branch-and-bound solver. Additionally, due to numerical difficulties, some branch-and-bound nodes may only be processable by an exact LP solver.

To summarize, the pure rational approach is always applicable but introduces a large overhead in running time while the safe-FP approach is more efficient but of limited applicability.

Since we want to solve MIPs that are given by rational data efficiently and exactly we have developed a version of branch-and-bound that attempts to combine the advantages of the pure rational and safe-FP approaches, and to compensate for their individual weaknesses. The idea is to work with two branch-and-bound processes. The *main process* implements the rational approach. Its result is surely correct and will be issued to the user. The other one serves as a *slave process*, where the faster safe-FP approach is applied. To achieve reasonable running time, whenever possible the expensive rational computation of the main process will be skipped and certain decisions from the faster safe-FP process will be substituted. In particular, safe dual bound computations in the slave process can often replace exact LP solves in the main process. The rational process provides the exact problem data, allows to correctly store primal solutions, and makes exact LP solves possible whenever needed.

Algorithm 1 Branch-and-bound for exactly solving MIPs

Input: (MIP) $\max\{c^T x : x \in P\}$ with $P := \{x \in \mathbb{R}^n : Ax \leq b, x_i \in \mathbb{Z} \text{ for all } i \in I\}$, $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $I \subseteq \{1, \ldots, n\}$.

Output: *Exact* optimal solution $x^\star$ of MIP with objective value $c^\star$ or conclusion that MIP is infeasible ($c^\star = -\infty$).

1. *FP-problem:* Store (FP-MIP) $\max\{\widetilde{c}^T x : x \in \widetilde{P}\}$ with $\widetilde{P} := \{x \in \mathbb{R}^n : \widetilde{A}x \leq \widetilde{b}, x_i \in \mathbb{Z} \text{ for all } i \in I\}$, $\widetilde{A} \in \mathbb{M}^{m \times n}$, $\widetilde{b} \in \mathbb{M}^m$, and $\widetilde{c} \in \mathbb{M}^n$.
2. *Init:* Set $\mathcal{L} := \{(P, \widetilde{P})\}$, $L := -\infty$, x^{MIP} to be empty, and $c^{\text{MIP}} := -\infty$.
3. *Abort:* If $\mathcal{L} = \emptyset$, stop and return x^{MIP} and c^{MIP}.
4. *Node selection:* Choose $(P_j, \widetilde{P}_j) \in \mathcal{L}$ and set $\mathcal{L} := \mathcal{L} \setminus \{(P_j, \widetilde{P}_j)\}$.
5. *Dual bound:* Solve LP-relaxation $\max\{\widetilde{c}^T x : x \in \widetilde{LP}_j\}$ *approximately.*
   (a) If $\widetilde{LP}_j$ is *claimed* to be empty, *safely* check if LP_j is empty.
      i. If LP_j is empty, set $c^\star := -\infty$.
      ii. If LP_j is not empty, solve LP-relaxation $\max\{c^T x : x \in LP_j\}$ *exactly.* Let $x^\star$ be an *exact* optimal LP solution and $c^\star$ its objective value.
   (b) If $\widetilde{LP}_j$ is *claimed* not to be empty, let $x^\star$ be an *approximate* optimal LP solution and compute a *safe* dual bound $c^\star$ with $\max\{c^T x : x \in LP_j\} \leq c^\star$.
6. *Bounding:* If $\overline{c^\star} \leq L$, goto Step 3.
7. *Primal bound:*
   (a) If $x^\star$ is *approximate* LP solution and claimed to be feasible for FP-MIP, solve LP-relaxation $\max\{c^T x : x \in LP_j\}$ *exactly.* If LP_j is *in fact* empty, goto Step 3. Otherwise, let $x^\star$ be an *exact* optimal LP solution and $c^\star$ its objective value.
   (b) If $x^\star$ is *exact* LP solution and *truly* feasible for MIP:
      i. If $c^\star > c^{\text{MIP}}$, set $x^{\text{MIP}} := x^\star$, $c^{\text{MIP}} := c^\star$, and $L := \underline{c^\star}$.
      ii. Goto Step 3.
8. *Branching:* Choose index $i \in I$ with $x_i^\star \notin \mathbb{Z}$.
   (a) Split P_j in $Q_1 := P_j \cap \{x : x_i \leq \lfloor x_i^\star \rfloor\}$, $Q_2 := P_j \cap \{x : x_i \geq \lceil x_i^\star \rceil\}$.
   (b) Split $\widetilde{P}_j$ in $\widetilde{Q}_1 := \widetilde{P}_j \cap \{x : x_i \leq \lfloor x_i^\star \rfloor\}$, $\widetilde{Q}_2 := \widetilde{P}_j \cap \{x : x_i \geq \lceil x_i^\star \rceil\}$.
   Set $\mathcal{L} := \mathcal{L} \cup \{(Q_1, \widetilde{Q}_1), (Q_2, \widetilde{Q}_2)\}$ and goto Step 3.

The complete procedure is given in Alg. 1. The set of FP-representable numbers is denoted by $\mathbb{M}$; lower and upper approximations of $x \in \mathbb{Q}$ are denoted $\underline{x} \in \mathbb{M}$ and $\overline{x} \in \mathbb{M}$, respectively. The slave process, which utilizes the safe-FP approach, works on a MIP instance with FP-representable data. It is set up in Step 1 of the algorithm. If the input data are already FP-representable, both processes solve the same MIP instance, i.e., $\widetilde{P} := P$ and $\widetilde{c} := c$ in Step 1. In principle, this results in employing only the safe-FP approach except for some necessary exact LP solves. Otherwise, an approximation of the MIP with $P \approx \widetilde{P}$, $c \approx \widetilde{c}$ or a relaxation with $P \subseteq \widetilde{P}$, $c = \widetilde{c}$ is constructed; called *FP-approximation* and *FP-relaxation*, respectively. The choice depends on the dual bounding method applied in the slave process (see Sect. 3).

On the implementation side, we maintain only a single branch-and-bound tree. At the root node of this common tree, we store the LP relaxations of both processes: $\max\{c^T x : x \in LP\}$ and $\max\{\widetilde{c}^T x : x \in \widetilde{LP}\}$. In addition, for each node, we know the branching constraint that was added to create the subproblem in both processes. Branching on variables, performed in Step 8, introduces the same bounds for both processes.

The use of primal and dual bounds to discard subproblems (see Steps 5, 6, and 7) is a central component of the branch-and-bound algorithm. In particular, in the exact MIP setting, the efficiency strongly depends on the strength of the dual bounds and the time spent generating them (Step 5). The starting point of this step is the *approximate LP solution* of the slave process. It is obtained by an LP solver that works on FP-numbers and accepts rounding errors; referred to as *inexact LP solver*. Depending on the result, we safely check whether the rational LP, i.e., the exact LP relaxation, is also infeasible or we compute a safe dual bound by post-processing the approximate LP solution. Different techniques are discussed in Sect. 3 and are computationally evaluated in Sect. 4. We only perform the exact but expensive dual bound computation of the main process if it is necessary (Step 5(a)ii).

Dual and primal bounds are stored as FP-numbers and the bounding in Step 6 is performed without tolerances; a computed bound that is not FP-representable is relaxed in order to be safe. For the primal (lower) bound L, this means $L < c^{\text{MIP}}$ if the objective value c^{MIP} of the incumbent solution x^{MIP} is not in $\mathbb{M}$.

Algorithm 1 identifies primal solutions by checking LP solutions for integrality. This check, performed in Step 7, depends on whether the LP was already solved exactly at the current node. If so, we exactly test the integrality of the exact LP solution. Otherwise, we decide if it is worth solving the LP exactly. We deem it worthy if the approximate LP solution is nearly integral. In this case, we solve the LP exactly, using the corresponding basis to warm start the exact LP solver (hopefully with few pivots and no need to increase the precision) and perform the exact integrality test on the exact LP solution. In order to correctly report the optimal solution found at the end of Step 3, the incumbent solution (that is, the best feasible MIP solution found thus far) and its objective value are stored as rational numbers.

This hybrid approach can be extended to a branch-and-cut algorithm with primal heuristics and presolving; but the focus of this article is on the development of the basic branch-and-bound framework.

3 Safe Dual Bound Generation Techniques

This section describes several methods for computing valid LP dual bounds. The overall speed of the MIP solver will be influenced by several aspects of the dual bounding strategy; how generally applicable the method is, how quickly the bounds can be computed and how strong the bounds are, because weak bounds can increase the node count.

3.1 Exact LP Solutions

The most straightforward way to compute valid LP bounds is to solve each node LP relaxation exactly. This strategy is always applicable and produces the tightest possible bounds. Within a branch-and-bound framework the dual simplex algorithm can be warm started with the final basis computed at the parent node, speeding up the LP solution process. The fastest exact rational LP solver currently available is QSOPT_EX [7]. The strategy used by this solver can be summarized as follows: the basis returned by a double-precision LP solver is tested for optimality by symbolically computing the corresponding basic solution, if it is suboptimal then additional simplex pivots are performed with an increased level of precision and this process is repeated until the optimal basis is identified. This method is considerably faster than using rational arithmetic exclusively and is usually no more than two to five times slower than inexact LP solvers. For problems where the basis determined by the double-precision subroutines of QSOPT_EX is not optimal, the additional increased precision simplex pivots and additional exact basic solution computations significantly increase the solution time.

3.2 Basis Verification

Any exactly feasible dual solution provides a valid dual bound. Therefore, valid dual bounds can be determined by symbolically computing the dual solution corresponding to a numerically obtained LP basis, without performing the extra steps required to identify the exact optimal solution. If the dual solution is feasible, its objective value gives a valid bound. If it is infeasible, an infinite bound is returned. Within the branch-and-bound algorithm, an infinite dual bound will lead to more branching. Due to the fixing of variables, branching often remediates numerical problems in the LP relaxations. This strategy avoids the extended precision simplex pivoting that can occur when solving each node LP exactly, but it can increase the number of nodes.

3.3 Primal-Bound-Shift

Valid bounds can also be produced by correcting approximate dual solutions. A special case occurs when all primal variables have finite upper and lower bounds. The following technique was employed by Applegate et. al. in the CONCORDE software package [5] and is described more generally for MIPs by Neumaier and Shcherbina [19]. Consider a primal problem of the form $\max\{c^T x : Ax \leq b, 0 \leq x \leq u\}$ with dual $\min\{b^T y + u^T z : A^T y + z \geq c,\ y, z \geq 0\}$. Given an approximate dual solution $\tilde{y}, \tilde{z}$, an exactly feasible dual solution $\hat{y}, \hat{z}$ is constructed by setting $\hat{y}_i = \max\{0, \tilde{y}_i\}$ and $\hat{z}_i = \max\{0, (c - A^T \hat{y})_i\}$. This gives the valid dual bound $b^T \hat{y} + u^T \hat{z}$. When working with a FP-relaxation of the original problem, this bound can be computed using floating-point arithmetic with safe directed rounding to avoid the symbolic computation of the dual feasible solution. The simplicity of computing this bound suggests it will be an excellent choice when

applicable. However, if some primal variable bounds are large or missing it may produce weak or infinite bounds, depending on the feasibility of $\tilde{y}, \tilde{z}$.

3.4 Project-and-Shift

Correcting an approximate dual solution to be exactly feasible in the absence of primal variable bounds is still possible. Consider a primal problem of the form $\max\{c^T x : Ax \leq b\}$ with dual $\min\{b^T y : A^T y = c,\ y \geq 0\}$. An approximate dual solution $\tilde{y}$ can be corrected to be feasible by projecting it into the affine hull of the dual feasible region and then shifting it to satisfy all of the non-negativity constraints, while maintaining feasibility of the equality constraints. These operations could involve significant computation if performed on a single LP. However, under some assumptions, the most time consuming computations can be performed only once at the root node of the branch-and-bound tree and reused for each node bound computation. The root node computations involve solving an auxiliary LP exactly and symbolically LU factoring a matrix; the cost of each node bound computation is dominated by performing a back-solve of a pre-computed symbolic LU factorization, which is often faster than solving a node LP exactly. This method is more generally applicable than the primal-bound-shift method, but relies on some conditions that are met by most, but not all, of the problems in our test set. A detailed description and computational study of this algorithm can be found in [20]. A related method is also described by Althaus and Dumitriu [4].

3.5 Combinations and Beyond

The ideal dual bounding method is generally applicable, produces tight bounds, and computes them quickly. Each of the four methods described so far represents some trade-off between these conflicting characteristics. The exact LP method is always applicable and produces the tightest possible bounds, but is computationally expensive. The primal-bound-shift method computes valid bounds very quickly, but relies on problem structure that may not always be present. The basis verification and project-and-shift methods provide compromises in between, with respect to speed and generality. Also, since the relative performance of these dual bounding methods strongly depends on the (sub)problem structure it may change throughout the tree. Therefore, a strategy that combines and switches between the bounding techniques is the best choice for an exact MIP solver intended to efficiently solve a broad class of problems.

In Sect. 4, we will evaluate the performance of each dual bounding method presented here and analyze in what situations which technique works best. In a final step, we then study different strategies to automatically decide how to compute safe dual bounds for a given MIP instance. The central idea is to apply fast primal-bound-shift as often as possible and if necessary employ another method depending on the problem structure. In this connection, we will address the question of whether this decision should be static or dynamic.

In the first version ("Auto"), we decide on the method dynamically in Step 5. At each node primal-bound-shift is applied, and in case it produces an infinite bound one of the other methods is applied. The drawbacks are that it allows for unnecessary computations and that it requires an FP-relaxation for the slave process in order to support primal-bound-shift. Alternatively, we can guess whether primal-bound-shift will work ("Auto-Static"). Meaning the dual bounding method is selected depending on the problem structure at the beginning, in Step 1, and remains fixed throughout the tree. This allows us to work with FP-approximations whenever we do not select primal-bound-shift.

Beyond that, we will analyze whether it is a good idea to compute safe dual bounds only if they are really required, i.e., at nodes where the unsafe bound would lead to pruning ("Auto-Limited"). Furthermore, we experiment with interleaving our actual selection strategy with exact LP solves to eliminate special cases where weak bounds cause the solver to keep branching in subtrees that would have been cut off by the exact LP bound ("Auto-Ileaved").

4 Computational Study

In this section, we investigate the performance of our exact MIP framework employing the different safe dual bounding techniques discussed above: primal-bound-shift ("BoundShift"), project-and-shift ("ProjectShift"), basis verification ("VerifyBasis"), and exact LP solutions ("ExactLP"). We will first look at each method on its own and then examine them within the branch-and-bound algorithm. At the end, we discuss and test strategies to automatically switch between the most promising bounding techniques.

The discussed algorithms were implemented into the branch-and-bound algorithm provided by the MIP framework SCIP 1.2.0.8 [1,2,23], using best bound search for node selection and first fractional variable branching. To solve LPs approximately and exactly we call Cplex 12.2 [15] and QSopt_ex 2.5.5 [6], respectively. Rational computations are based on the GMP library 4.3.1 [13]. All benchmark runs were conducted on 2.5 GHz Intel Xeon E5420 CPUs with 4 cores and 16 GB RAM each. To maintain accurate results only one computation was run at the same time. We imposed a time limit of 24 hours and a memory limit of 13 GB. The timings used to measure computation times are always rounded up to one second if they are smaller.

Our test set contains all instances of the Miplib 3.0 [8] and Miplib 2003 [3] libraries and from the Mittelmann collections [18] that can be solved within 2 hours by the inexact branch-and-bound version of SCIP ("Inexact"). This gives a test suite of 57 MIP instances (30:70:4_5:0_95:100, acc-0, acc-1, acc-2, air03, air05, bc1, bell3a, bell5, bienst1, bienst2, blend2, dano3_3, dano3_4, dano3_5, dcmulti, egout, eilD76, enigma, flugpl, gen, gesa3, gesa3_o, irp, khb05250, l152lav, lseu, markshare1_1, markshare4_0, mas76, mas284, misc03, misc07, mod008, mod010, mod011, neos5, neos8, neos11, neos21, neos897005, nug08, nw04, p0033, p0201, pk1, qap10, qnet1_o, ran13x13, rentacar, rgn, stein27, stein45, swath1, swath2, vpm1, vpm2).

Table 1. Root node dual bound: Relative difference to "ExactLP" dual bound and additional computation time "DB" in geometric mean.

Setting	Zero	S	M	L	∞	DB (s)
BoundShift	13	26	2	0	16	1.0
ProjectShift	19	31	5	0	2	2.8
VerifyBasis	57	0	0	0	0	1.3
ExactLP	57	0	0	0	0	1.4
Auto	20	35	2	0	0	1.3
Auto-Static	21	34	2	0	0	1.3
Auto-Ileaved	20	35	2	0	0	1.3

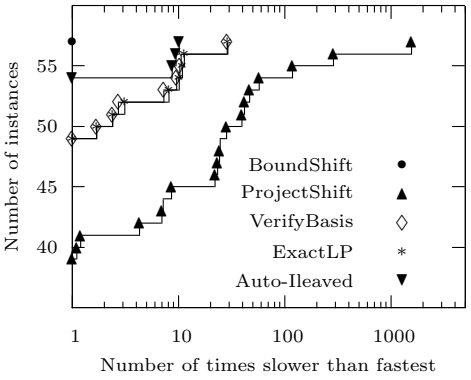

Figure 1. Comparison of safe dual bounding times "DB" at root.

Note that we also analyzed the performance on the other, harder, instances of the libraries by looking at the final gap and the number of branch-and-bound nodes processed within a certain time limit. The conclusions drawn here, on the smaller suite, were confirmed by these results.

4.1 Root Node Performance

We start with evaluating the root node behavior of the dual bounding methods. Our performance measures are: time overhead and bound quality. The performance profile, see [11], in Fig. 1 visualizes the relative overhead times for the safe dual bounding methods. For each of them, it plots the number of instances for which the safe dual bounding step was performed within a given factor of the bounding time of the fastest method. Table 1 presents the geometric mean of these additional safe dual bounding times in seconds ("DB") and states the number of instances for which a certain dual bound quality was achieved.

This quality is given by the relative difference between the computed safe dual bound $c^\star$ and the exact LP value $c^{\star\star} := \max\{c^T x : x \in LP_j\}$. However, we actually compare the FP-representable upper approximations of both values, as used in Alg. 1, and define the relative difference as $d := (\overline{c^\star} - \overline{c^{\star\star}})/\max\{1, |\overline{c^\star}|, |\overline{c^{\star\star}}|\}$. The corresponding columns in Table 1 are: "Zero" difference for $d = 0$, "S(mall)" difference for $d \in (0, 10^{-9}]$, "M(edium)" difference for $d \in (10^{-9}, 10^{-3}]$, and "L(arge)" difference for $d \in (10^{-3}, \infty)$. Column "∞" counts the worst case behavior, i.e., infinite dual bounds.

We observe that basis verification has similar behavior as exact LP for the root node. However, as we will see in the next section, it will give an improvement over the exact LP solver when expensive basis repair steps are required to find the exact solution.

As expected, primal-bound-shift is the fastest method. However, it produces infinite dual bounds on 16 instances in contrast to only 2 fails for project-and-shift and no fails for basis verification. This is, the bases obtained by CPLEX are

Table 2. Overall performance. "slv" is number of instances solved, "DB" is safe dual bounding time.

		Geometric mean for instances solved by all settings (37)		
Setting	slv	Nodes	Time (s)	DB (s)
Inexact	57	18 030	59.4	—
BoundShift	43	24 994	110.4	13.9
ProjectShift	49	18 206	369.3	238.1
VerifyBasis	51	18 078	461.8	329.8
ExactLP	51	18 076	550.7	419.0
Auto	54	18 276	92.6	17.5
Auto-Static	53	18 276	100.2	19.8
Auto-Ileaved	55	18 226	91.4	18.4
Auto-Limited	48	22 035	89.9	12.0

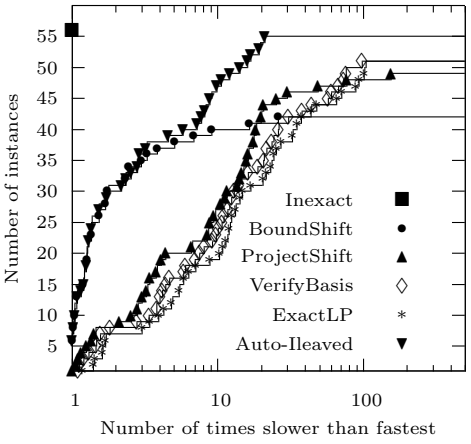

Figure 2. Comparison of overall solving times "Time".

often dual feasible and even optimal and project-and-shift meets its requirements most of the time. Still, the finite bounds provided by primal-bound-shift are of very good quality; most of them fall into the "Zero" and "S(mall)" categories. Thus, when primal-bound-shift works we expect to obtain strong bounds and whenever it fails we assume basis verification or project-and-shift to be applicable.

Where basis verification is in most cases only up to 10 times slower than primal-bound-shift, project-and-shift is up to 100 times slower at the root node because of the expensive initial set-up step. However, as we will see, the overhead incurred by the set-up step of project-and-shift often pays off when it is applied within the entire branch-and-bound tree.

4.2 Overall Performance

We will now analyze the effect of the dual bound methods on the overall performance of the exact MIP solver and compare it with the inexact branch-and-bound version of SCIP ("Inexact"). Table 2, reports the number of instances that were solved within the imposed limits (Column "slv"), for each setting. On 37 instances, all settings succeeded. For this group ("all solved"), we present in Table 2, the number of branch-and-bound nodes "Nodes", the solution time "Time" in seconds, and the additional time spent in the safe dual bounding step "DB" in seconds, all in geometric mean for each method. In addition, Fig. 2 gives a performance profile comparing the solution times. For a setting where an instance had a timeout, it is reported with an infinite ratio to the fastest setting. Thus, the intersection points at the right boarder of the graphic reflect the "slv" column.

The observations made for the root node carry forward to the application in the branch-and-bound algorithm. Primal-bound-shift leads to the fastest node

processing. Basis verification has a slightly better performance than solving LPs exactly. However, basis verification is often outperformed by project-and-shift.

A closer look reveals that primal-bound-shift introduces not only a small overhead at the root but throughout the tree as well. For the individual instances it could solve, we usually experience a slow-down of at most 2. The few large slow-down factors of up to 10 can be explained by a node increase due to a small number of missing variable bounds and by expensive exact LP calls. However, due to its limited applicability we solved only 43 instances.

In contrast, project-and-shift solves 49 instances. That is, the often observed good dual bound quality at the root node seems to stay throughout the tree. We already pointed out the big overhead this costs at the root node. Nevertheless, the method is fast on the rest of the tree and leads to an acceptable slow-down compared to the inexact code. In mean, it is only 6 times slower on the "all solved" group. In this fashion, most of the instances solved by this setting are only up to 20 times slower than the inexact code. If we compare project-and-shift with basis verification we see a similar and often better performance for project-and-shift. Still, on some instances basis verification works better. For example, it solves two more instances of our test set. We examined different problem characteristics and found the number of non-zeros in the constraint matrix to be a good criterion for choosing between project-and-shift and basis verification. In the automatic dual bound selection strategies, we prefer project-and-shift as long as the matrix has at most 10,000 non-zeros.

4.3 Combinations

We already gave some remarks concerning a strategy that automatically chooses a dual bounding method. Another important observation for this purpose is that replacing FP-approximations by FP-relaxations does not affect the performance much: running project-and-shift on an FP-relaxation gave similar results on our test set. Therefore, we decided to always set up an FP-relaxation in Step 1. This way, we are allowed to apply primal-bound-shift at any node we want to.

The automatic decision process used in the "Auto" run works as follows. At every node, we first test whether primal-bound-shift produces a finite bound. If not, we choose project-and-shift or basis verification depending on the structure of the constraint matrix as explained above. The root node results for the combined versions are presented in Table 1 and Fig. 1; the overall performance results can be found in Table 2 and Fig. 2. Note that we excluded "Auto-Limited" from Table 1 as it never computed safe finite bounds at the root node and that we only included the best auto setting in the performance profiles as their graphs look very similar.

The experiments show that "Auto" actually combines the advantages of all dual bounding methods. We can solve all 43 instances that primal-bound-shift solved as well as 11 additional ones by automatically switching to other dual bounding methods at the nodes.

In Sect. 3.5, we discussed three possible improvements for the automatic dual bound selection procedure. The first one, to only guess whether primal-bound-

shift will work, is implemented in the test run "Auto-Static". The guess is static, i.e., does not change throughout the tree; we skip primal-bound-shift if more than 20% of the problem variables have lower or upper bounds with absolute value larger than 10^6. Comparing both automatic settings shows that it is no problem to actually test at each node whether primal-bound-shift works, and it even leads to a slightly improved performance.

The second idea was to interleave the strategy with exact LP calls whenever a node is very likely to be cut off ("Auto-Ileaved"). This does not apply often, but is helpful when it does. We solved one more instance to optimality without introducing a significant time overhead on the other instances. The third extension was to only compute bounds safely if they are actually used for a crucial decision, i.e., if the unsafe bound with tolerances would lead to cutting off a node. Looking at the overall behavior for the corresponding test run "Auto-Limited", it is not clear whether this is a good idea in general. It only solved 48 instead of 54 instances. On the other hand, we experienced that on harder instances the node count at timeout strongly increases, i.e., the node processing is much faster on average. However, we cannot draw any conclusions about the quality of this approach on harder instances as in this setting, the primal-dual-gap does not improve steadily. Further testing is needed here, e.g., by applying additional safe dual bounding steps at selected levels of the tree.

5 Conclusion

From the computational results we can make several key observations. Each dual bounding method studied has strengths and weaknesses depending on the problem structure. Automatically switching between these methods in a smart way solves more problems than any single dual bounding method on its own. Of the 57 problems solved within two hours by the floating-point branch-and-bound solver, 55 could also be solved exactly within 24 hours and the solution time was usually no more than 10 times slower. This demonstrates that the hybrid methodology can lead to an efficient exact branch-and-bound solver, not limited to specific classes of problems. As our focus has been exclusively on the branch-and-bound procedure, we have compared the exact solver against a floating-point solver restricted to pure branch-and-bound. The exact solver is still not directly competitive with the full version of SCIP with its additional features enabled. However, it is realistic to think that the future inclusion of additional MIP machinery such as cutting planes, presolving, and primal heuristics into this exact framework could lead to a full featured exact MIP solver that is not prohibitively slower than its inexact counterparts.

Acknowledgments

The authors would like to thank Tobias Achterberg for helpful discussions on how to best incorporate these features into SCIP. We would also like to thank

Daniel Espinoza for his assistance with QSopt_ex, which included adding new features and writing an interface for use within SCIP.

References

1. Achterberg, T.: Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin (2007)
2. Achterberg, T.: SCIP: Solving constraint integer programs. Math. Programming Computation 1(1), 1–41 (2009)
3. Achterberg, T., Koch, T., Martin, A.: The mixed integer programming library: MIPLIB 2003. http://miplib.zib.de
4. Althaus, E., Dumitriu, D.: Fast and accurate bounds on linear programs. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 40–50. Springer (2009)
5. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (2006)
6. Applegate, D.L., Cook, W.J., Dash, S., Espinoza, D.G.: QSopt_ex. http://www.dii.uchile.cl/~daespino/ESolver_doc/main.html
7. Applegate, D.L., Cook, W.J., Dash, S., Espinoza, D.G.: Exact solutions to linear programming problems. Oper. Res. Lett. 35(6), 693–699 (2007)
8. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.: An updated mixed integer programming library: MIPLIB 3.0. Optima 58, 12–15 (1998)
9. Cook, W.J., Dash, S., Fukasawa, R., Goycoolea, M.: Numerically safe Gomory mixed-integer cuts. INFORMS J. Comput. 21(4), 641–649 (2009)
10. Dhiflaoui, M., Funke, S., Kwappik, C., Mehlhorn, K., Seel, M., Schömer, E., Schulte, R., Weber, D.: Certifying and repairing solutions to large LPs, how good are LP-solvers? In: SODA 2003. pp. 255–256. ACM/SIAM (2003)
11. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Programming 91(2), 201–213 (2001)
12. Espinoza, D.G.: On Linear Programming, Integer Programming and Cutting Planes. Ph.D. thesis, Georgia Institute of Technology (2006)
13. GMP: GNU multiple precision arithmetic library. http://gmplib.org
14. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys (CSUR) 23(1), 5–48 (1991)
15. IBM ILOG: CPLEX. http://www.ilog.com/products/cplex
16. Koch, T.: The final NETLIB-LP results. Oper. Res. Lett. 32,(2), 138–142 (2004)
17. Kwappik, C.: Exact Linear Programming. Master thesis, Universität des Saarlandes (1998)
18. Mittelmann, H.D.: Benchmarks for Optimization Software. http://plato.asu.edu/bench.html (2010)
19. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Programming 99(2), 283–296 (2004)
20. Steffy, D.E.: Topics in Exact Precision Mathematical Programming. Ph.D. thesis, Georgia Institute of Technology (2011)
21. de Vries, S., Vohra, R.: Combinatorial Auctions: A Survey. INFORMS J. Comput. 15(3), 284–309 (2003)
22. Wilken, K., Liu, J., Heffernan, M.: Optimal instruction scheduling using integer programming. SIGPLAN Notices 35(5), 121–133 (2000)
23. Zuse Institute Berlin: SCIP. http://scip.zib.de