Improving the Accuracy of Linear Programming Solvers with Iterative Refinement

Ambros M. Gleixner
Zuse Institute Berlin
Berlin, Germany
gleixner@zib.de

Daniel E. Steffy
Mathematics and Statistics
Oakland University
Rochester, Michigan, USA
steffy@oakland.edu

Kati Wolter
Zuse Institute Berlin
Berlin, Germany
wolter@zib.de

ABSTRACT

We describe an iterative refinement procedure for computing extended precision or exact solutions to linear programming problems (LPs). Arbitrarily precise solutions can be computed by solving a sequence of closely related LPs with limited precision arithmetic. The LPs solved share the same constraint matrix as the original problem instance and are transformed only by modification of the objective function, right-hand side, and variable bounds. Exact computation is used to compute and store the exact representation of the transformed problems, while numeric computation is used for solving LPs. At all steps of the algorithm the LP bases encountered in the transformed problems correspond directly to LP bases in the original problem description.

We demonstrate that this algorithm is effective in practice for computing extended precision solutions and that this leads to direct improvement of the best known methods for solving LPs exactly over the rational numbers.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; G.4 [**Mathematics of Computing**]: Algorithm design and analysis; I.1.2 [**Computing Methodologies**]: Symbolic and algebraic manipulation—*Performance evaluation of algorithms and systems*

General Terms

Algorithms, Performance, Optimization

Keywords

Linear programming, iterative refinement

1. INTRODUCTION

Most computer codes available today for solving linear programs (LPs) use floating-point arithmetic, which can lead to numerical errors. Although such codes are effective at computing approximate solutions for a wide range of instances, there are situations when their results are unreliable, or when extended precision or exact solutions are desirable. Fast algorithms for exact linear programming are also directly useful as subroutines for solving mixed-integer programming problems exactly [1, 7].

Some recent articles that have used the exact solution of linear or integer programming instances to establish theoretical results include [3, 4, 5, 9, 14, 15, 16]. Computational tools for exact linear and integer programming have not been readily available until recently. Improving their speed and capabilities will expand the range of problems and instance sizes where they can be successfully applied.

The purpose of this article is to develop and study a technique to build extended precision LP solutions using an approximate LP solver as a subroutine. As a byproduct, this algorithm can also be used to solve LPs exactly over the rational numbers faster than previous methods.

ISSAC '12

2. BACKGROUND

2.1 Linear programming from a simplex perspective

Linear programming has become one of the most essential tools in theory and practice of mathematical optimization due to strong duality theory and the availability of fast solution algorithms. In this section, we summarize some well-known facts and sketch the idea of the simplex algorithm, one of the most commonly used solution methods. More details can be found in many textbooks such as [6]. For readers familiar with the simplex algorithm, this will only serve as an introduction to our notation.

A linear program consists of a linear objective function, linear constraints, and bounds on the variables. It can be written in the form

$$\min\{c^\mathsf{T} x : Ax = b, \ell \leqslant x \leqslant u\}$$

with objective function vector $c \in \mathbb{R}^n$, bounds $\ell \in (\mathbb{R} \cup \{-\infty\})^n$ and $u \in (\mathbb{R} \cup \{+\infty\})^n$, constraint matrix $A \in \mathbb{R}^{m \times n}$, and right-hand side $b \in \mathbb{R}^m$. This form is often called *computational form*, since it is used by most implementations of the simplex algorithm. Inequality constraints $L \leqslant a^\mathsf{T} x \leqslant U$ can be converted to computational form by introducing an auxiliary slack variable, $a^\mathsf{T} x - s = 0$, $L \leqslant s \leqslant U$. W.l.o.g. we may assume that A has full rank and $n \geqslant m$.

For the sake of clarity, our presentation will assume zero

lower bounds and infinite upper bounds for all variables,

$$\min\{c^\mathsf{T}x : Ax = b, x \geqslant 0\}, \tag{1}$$

however, the methods proposed are easily implemented for general LPs.

The feasible region $F = \{x \in \mathbb{R}^n : Ax = b, x \geqslant 0\}$ of (1) is a polyhedron. Problem (1) is called *infeasible* if F is empty, *unbounded* if it contains solutions with arbitrarily low objective value $c^\mathsf{T}x$, but unless stated otherwise, we will assume that an *optimal* solution exists, i.e., an $x^* \in F$ with $c^\mathsf{T}x^* \leqslant c^\mathsf{T}x$ for all $x \in F$. Importantly, if an optimal solution exists then there also exists an optimal vertex of F.

The vertices of F are a special case of basic solutions. Basic solutions are solutions uniquely determined by $n - m$ variables held fixed at their bounds. The so-called *basis* $\mathcal{B} \subseteq \{1, \ldots, n\}$, $|\mathcal{B}| = m$, is formed by the indices of the remaining, unfixed variables. This transforms the constraints $Ax = b$ to

$$A_\mathcal{B}x_\mathcal{B} = b, \tag{2}$$

where $A_\mathcal{B} \in \mathbb{R}^{m\times m}$ denotes the submatrix of A formed by the columns in $\mathcal{B}$ and $x_\mathcal{B}$ denotes the vector of basic variables x_i, $i \in \mathcal{B}$. In the following, we consider only bases with regular $A_\mathcal{B}$. Then (2) uniquely defines a primal solution $x_\mathcal{B} = A_\mathcal{B}^{-1}b$, $x_i = 0$ for $i \notin \mathcal{B}$. Furthermore, the system

$$A_\mathcal{B}{}^\mathsf{T}y = c_\mathcal{B} \tag{3}$$

provides us with a unique dual solution $y \in \mathbb{R}^m$. This way, each variable is associated with its *reduced cost* $c_i - A_i^\mathsf{T}y$, $i \in \{1, \ldots, n\}$, where A_i denotes the i-th column of A. Note that basic solutions satisfy the *complementary slackness* conditions $x_i = 0 \vee A_i^\mathsf{T}y = c_i$ for all $i \in \{1, \ldots, n\}$.

For a basis $\mathcal{B}$, let x^*, y^* be the corresponding primal-dual solution. By definition, x^* satisfies the equality constraints, but may violate the nonnegativity constraints. We call $\mathcal{B}$ *primal feasible* if $x_\mathcal{B} \geqslant 0$ and *dual feasible* if the reduced costs are nonnegative, i.e., $c_i - A_i^\mathsf{T}y \geqslant 0$ for all $i \notin \mathcal{B}$. The vertices of F are precisely the primal feasible basic solutions. Dual feasibility certifies the optimality of x^* since

$$c^\mathsf{T}x \geqslant (A^\mathsf{T}y^*)^\mathsf{T}x \overset{(1)}{=} y^{*\mathsf{T}}b \overset{(2)}{=} y^{*\mathsf{T}}A_\mathcal{B}x^*_\mathcal{B} \overset{(3)}{=} c_\mathcal{B}{}^\mathsf{T}x^*_\mathcal{B} = c^\mathsf{T}x^*$$

for all $x \in F$. Hence, a primal and dual feasible basis is called optimal.

Note that basic solutions are discrete objects, and (1) could in theory be solved by enumerating its finitely many bases. More sophisticatedly, the *primal simplex algorithm* constructs a sequence of neighboring primal feasible bases, i.e., neighboring vertices of F, with increasing objective value until the reduced costs are all nonnegative, i.e., dual feasibility proves optimality.

At a primal feasible basis $\mathcal{B}$, the basic variables are a function of the nonbasic variables as can be seen by rewriting (2) in more detail as

$$x_\mathcal{B} = A_\mathcal{B}^{-1}(b - \sum_{i\notin\mathcal{B}} A_ix_i) = A_\mathcal{B}^{-1}b - \sum_{i\notin\mathcal{B}}(A_\mathcal{B}^{-1}A_i)x_i. \tag{4}$$

The neighboring bases of $\mathcal{B}$ are reached by increasing the value of a single nonbasic variable x_k , $k \notin \mathcal{B}$, from 0 up to the maximum value such that $x_\mathcal{B} = A_\mathcal{B}^{-1}b - (A_\mathcal{B}^{-1}A_k)x_k \geqslant 0$ still holds. One basic variable, x_ℓ, $\ell \in \mathcal{B}$, becomes tight at its bound and we have reached a new basis $\mathcal{B}' = (\mathcal{B}\setminus\{\ell\})\cup\{k\}$, which can be proven to be regular.

The reduced cost of the increased nonbasic variable x_k determines the change of the objective function value,

$$\begin{aligned} c^\mathsf{T}x = c_\mathcal{B}^\mathsf{T}x_\mathcal{B} + c_kx_k &\overset{(4)}{=} c_\mathcal{B}^\mathsf{T}A_\mathcal{B}^{-1}b - c_\mathcal{B}^\mathsf{T}(A_\mathcal{B}^{-1}A_k)x_k + c_kx_k \\ &\overset{(3)}{=} c_\mathcal{B}^\mathsf{T}A_\mathcal{B}^{-1}b + (c_k - A_k^\mathsf{T}y)x_k. \end{aligned}$$

Hence, by inspecting the reduced costs, the simplex algorithm can either conclude optimality or keep choosing a nonbasic variable with negative reduced cost to decrease the objective function value. Zero step length, i.e., when x_k enters the basis but its value remains at zero, may occur at so-called degenerate bases, but special techniques have been developed to nevertheless guarantee a convergent algorithm.

The *dual simplex algorithm* works analogously, only that it maintains a dual feasible basis and keeps choosing negative basic variables to leave the basis until $x_\mathcal{B} \geqslant 0$ is satisfied. Both variants must be preceded by a *phase one* algorithm to construct a primal respectively dual feasible starting basis. One technique employed, e.g., by the LP solver SoPlex [25, 29] used in our computational experiments, is to either modify the variable bounds or the objective function in order to artificially guarantee primal or dual feasibility, respectively, of a regular basis formed by slack variables. Then, primal or dual simplex is applied and yields an optimal solution which preserves its dual or primal feasibility, respectively, even after returning to the original problem.

Most of the computational effort is incurred by linear algebra routines involving the basis matrix $A_\mathcal{B}$ to solve (2) and (3), or to compute the step direction. State-of-the-art solvers heavily exploit sparsity of the input data and employ an LU factorization of $A_\mathcal{B}$ which is not recomputed at each iteration, but can be updated cheaply.

The most relevant alternatives to the simplex method in practice are interior point algorithms, for which polynomial running time can be proven. In contrast, the theoretical running time of the known simplex variants is exponential. Nevertheless, state-of-the-art implementations of the dual simplex algorithm are competitive with interior point codes for solving LPs from scratch and have the advantage of allowing for fast reoptimization after small modifications to the problem by warm starting from the preceding basis.

2.2 Exact methods for linear programming

A trivial method to solve LPs exactly over the rational numbers might be to apply a simplex algorithm and perform all computations in exact arithmetic. For all but few LP instances of interest, this idea is not viable. In the following, we will briefly present recent research on efficiently solving LPs exactly over the rational numbers.

These methods exploit the basis information provided by the simplex algorithm. If an optimal basis is identified, then an optimal primal-dual solution can be computed exactly using equations (2) and (3). Note that, when computed exactly, the primal-dual solution is a certificate of its own optimality, regardless of how its basis was obtained.

It has been observed that LP bases returned by floating-point solvers are often optimal for real world problems [10]. Koch [17] could compute optimal bases to all of the NETLIB LP instances [21] using only floating-point LP solvers and subsequently certifying them in exact rational arithmetic.

Applegate, Cook, Dash, and Espinoza [2] developed a simplex-based general purpose exact LP solver, QSopt_ex, that exploits this behavior to achieve fast computation times

on average and is capable of solving general LPs exactly over the rational numbers. If an optimal basis is not identified by the double-precision subroutines, more simplex pivots are performed using increased levels of precision until the exact rational solution is identified. A simplified version of this procedure is summarized as Algorithm 1. Whenever possible, the LP solve in line 4 is warm started with a basis $\mathcal{B}$ computed at a previous iteration. Analogous ideas are used for infeasible and unbounded LPs, see [11].

Algorithm 1 Incremental precision boosting for exact LP

1: **input:** $\min\{c^\mathsf{T}x : Ax = b, x \geqslant 0\}$ in rational precision
2: **for** precision = double, 128, 256, ..., rational **do**
3: get $\bar{A}, \bar{b}, \bar{c} \approx A, b, c$ in current precision
4: solve $\min\{\bar{c}^\mathsf{T}x : \bar{A}x = \bar{b}, x \geqslant 0\}$ in current precision
5: get basis $\mathcal{B}$ returned as optimal
6: compute exact rational primal-dual solution for $\mathcal{B}$
7: **if** exact primal-dual solution is optimal **then**
8: break
9: **end if**
10: **end for**
11: **return:** rational primal-dual solution, basis $\mathcal{B}$

QSopt_ex is often very effective at finding exact solutions quickly, especially when the double-precision LP subroutines are able to find an optimal LP basis. However, in cases that extended precision computations are used to identify the optimal basis, or when the rational systems of equations solved to compute the rational solution are difficult, solution times can increase significantly. We will refer to this strategy of iteratively increasing the working precision for the simplex algorithm as *incremental precision boosting*.

We also refer the reader to [30] for a general discussion on Exact and Robust Computational Geometry; although it does not discuss exact linear programming directly, many of the ideas are of direct relevance.

2.3 Iterative refinement for linear systems of equations

Iterative refinement is a commonly applied technique for finding accurate solutions to linear systems of equations and can be summarized as follows: Given a system of linear equations $Ax = b$, a sequence of increasingly accurate solutions $\{x^0, x^1, \ldots\}$ is constructed by first computing an approximate solution x^0, with $Ax^0 \approx b$. Then for $i \geqslant 1$, a refined solution $x^i \leftarrow x^{i-1} + c^{i-1}$ is computed where c^{i-1} satisfies $Ac^{i-1} \approx r^{i-1}$ and is a correction of the error $r^{i-1} = b - Ax^{i-1}$ observed from the solution at the previous iteration. This procedure can either be applied in *fixed precision*, where all operations are performed using the same level of precision, or in *mixed precision* where the residual errors r^i are computed with a higher level of precision than the system solves to compute the corrections c^i. For more details, see [13, 28].

Iterative refinement can also be used as a subroutine for computing exact solutions to rational systems of linear equations. After a sufficiently accurate solution x^i has been constructed, the continued fraction method can be used to reconstruct the exact solution; this strategy is guaranteed to work as long as x^i satisfies an accuracy threshold that can be determined a priori. This idea was described by Ursic and Patarra [26] and improved upon by Wan [27]. For additional recent developments see [23, 24].

There are a number of ways in which the idea of iterative refinement could be applied to solving linear programs. In [8], iterative refinement was investigated alongside other strategies for computing the exact rational solutions in line 6 of Algorithm 1; in many cases, this was faster than the other strategies tested. We also note that QSopt_ex does try to round the approximate primal-dual solution it has computed in line 4 to the exact rational solution using a continued fraction approximation before applying other methods to compute the exact basic solution from scratch in line 6.

One possible adjustment to Algorithm 1 would be to apply iterative refinement internally at several components of the simplex algorithm, as an alternative to increasing the working precision. Although such a strategy could yield an improvement we will take it one step further and instead solve a sequence of LPs, each one computing a correction of the previous, to build and refine an accurate primal-dual solution and corresponding basis. This strategy will simultaneously refine both the primal and dual solutions, by adjusting the primal feasible region and objective function of the LP to be solved.

3. ITERATIVE REFINEMENT LINEAR PROGRAMMING

3.1 The main idea

We now introduce our iterative refinement algorithm for linear programming. The main idea of the algorithm is as follows. First, the LP is solved approximately, producing a primal-dual solution x^*, y^*. Then, based on the error in x^*, y^*, a modified problem is created by shifting and scaling the primal and dual feasible regions of the original instance; a solution to this newly constructed problem gives a correction that is used to refine the accuracy of x^*, y^*. This process is iterated – repeatedly correcting the candidate solution – until it meets a required accuracy. The procedure is outlined in Algorithm 2. All operations are performed in exact rational arithmetic unless otherwise noted.

Before explaining Algorithm 2 in more detail, let us prove its correctness by showing that solving the transformed problem is equivalent to solving the original problem. As we will now see this holds for arbitrary vectors $\hat{x}, \hat{y}$.

Proposition 3.1. *Let $P = \min\{c^\mathsf{T}x : Ax = b, x \geqslant 0\}$ as in (1), let $\hat{x} \in \mathbb{R}^n$, $\hat{y} \in \mathbb{R}^m$ and scaling factors Δ_P, $\Delta_D > 0$ be given. Consider the transformed problem*

$$\hat{P} = \min\{\hat{c}^\mathsf{T}x : Ax = \hat{b}, x \geqslant -\Delta_P\hat{x}\}$$

with $\hat{c} = \Delta_D(c - A^\mathsf{T}\hat{y})$ and $\hat{b} = \Delta_P(b - A\hat{x})$. Let $x^* \in \mathbb{R}^n$, $y^* \in \mathbb{R}^m$, then the following hold:

1. *x^* is primal feasible for $\hat{P}$ within absolute tolerance $\epsilon_P \geqslant 0$ if and only if $\hat{x} + \frac{x^*}{\Delta_P}$ is primal feasible for P within ϵ_P/Δ_P.*
2. *y^* is dual feasible for $\hat{P}$ within absolute tolerance $\epsilon_D \geqslant 0$ if and only if $\hat{y} + \frac{y^*}{\Delta_D}$ is dual feasible for P within ϵ_D/Δ_D.*
3. *x^*, y^* satisfy the complementary slackness conditions for $\hat{P}$ if and only if $\hat{x} + \frac{x^*}{\Delta_P}$, $\hat{y} + \frac{y^*}{\Delta_D}$ are complementary slack for P.*

Algorithm 2 Iterative ref. for primal and dual feasible LP

1: **input:** $\min\{c^\mathsf{T}x : Ax = b, x \geqslant 0\}$ in rational precision, termination tolerances ϵ_P, ϵ_D, scaling limit α, iteration limit $k_{\max}$
2: `/* initial solve */`
3: $\Delta_P \leftarrow 1$, $\Delta_D \leftarrow 1$
4: solve $\min\{c^\mathsf{T}x : Ax = b, x \geqslant 0\}$ approximately
5: $\mathcal{B} \leftarrow$ basis returned as optimal
6: x^1, $y^1 \leftarrow$ primal-dual solution returned
7: **for** $k \leftarrow 1, 2, \ldots, k_{\max}$ **do**
8: `/* primal violation and scaling */`
9: $\hat{b} \leftarrow b - Ax^k$
10: $\delta_P \leftarrow \max\{\max_{j=1,\ldots,m} |\hat{b}_j|, \max_{i=1,\ldots,n} -x_i^k\}$
11: $\Delta_P \leftarrow \min\{1/\delta_P, \alpha\Delta_P\}$
12: `/* dual violation and scaling */`
13: $\hat{c} \leftarrow c - A^\mathsf{T}y^k$
14: $\delta_D \leftarrow \max\{\max_{i\in\mathcal{B}} |\hat{c}_i|, \max_{i\notin\mathcal{B}} -\hat{c}_i\}$
15: $\Delta_D \leftarrow \min\{1/\delta_D, \alpha\Delta_D\}$
16: **if** $\delta_P \leqslant \epsilon_P$ and $\delta_D \leqslant \epsilon_D$ **then**
17: break
18: **else**
19: `/* solve transformed problem */`
20: solve $\min\{\Delta_D\hat{c}^\mathsf{T}x : Ax = \Delta_P\hat{b}, x \geqslant -\Delta_P x^k\}$ approximately from starting basis $\mathcal{B}$
21: $\mathcal{B} \leftarrow$ basis returned as optimal
22: x^*, $y^* \leftarrow$ primal-dual solution returned
23: `/* perform correction */`
24: $x^{k+1} \leftarrow x^k + \frac{x^*}{\Delta_P}$
25: $y^{k+1} \leftarrow y^k + \frac{y^*}{\Delta_D}$
26: `/* force nonbasic variables to bounds */`
27: $x_i^{k+1} \leftarrow 0$ for all $i \notin \mathcal{B}$
28: **end if**
29: **end for**
30: **return:** x^k, y^k, basis $\mathcal{B}$

4. *x^*, y^* is an optimal primal-dual solution for $\hat{P}$ if and only if $\hat{x} + \frac{x^*}{\Delta_P}$, $\hat{y} + \frac{y^*}{\Delta_D}$ is optimal for P.*

5. *x^*, y^* is a basic primal-dual solution of $\hat{P}$ associated with basis $\mathcal{B}$ if and only if $\hat{x} + \frac{x^*}{\Delta_P}$, $\hat{y} + \frac{y^*}{\Delta_D}$ is a basic primal-dual solution for P associated with basis $\mathcal{B}$.*

Proof. For primal feasibility, point 1, we must check that the violation of variable bounds and equality constraints is simply scaled by $1/\Delta_P$,

$$\big(\hat{x} + \frac{x^*}{\Delta_P}\big) - 0 = \frac{x^* - (-\Delta_P\hat{x})}{\Delta_P}$$

and

$$A\big(\hat{x} + \frac{x^*}{\Delta_P}\big) - b = \frac{\Delta_P A\hat{x} + Ax^* - \Delta_P b}{\Delta_P} = \frac{Ax^* - \hat{b}}{\Delta_P}.$$

For dual feasibility, point 2, we check the reduced costs,

$$c - A^\mathsf{T}\big(\hat{y} + \frac{y^*}{\Delta_D}\big) = \frac{\Delta_D c - \Delta_D A^\mathsf{T}\hat{y} - A^\mathsf{T}y^*}{\Delta_D} = \frac{\hat{c} - A^\mathsf{T}y^*}{\Delta_D}.$$

This also shows that corresponding variable bounds and reduced costs are tight in P if and only if they are in $\hat{P}$, proving the claim on complementary slackness, point 3.

Since a solution is optimal if and only if it is primal and dual feasible and complementary slack, the first three points prove point 4. Finally, for a regular basis $\mathcal{B}$ we have

$$x_i^* = -\Delta_P\hat{x}_i \Leftrightarrow \hat{x}_i + \frac{x_i^*}{\Delta_P} = 0 \text{ for all } i \notin \mathcal{B},$$

$$A_\mathcal{B}x_\mathcal{B}^* = \hat{b} \Leftrightarrow A_\mathcal{B}\big(\hat{x} + \frac{x^*}{\Delta_P}\big) = b,$$

$$A_\mathcal{B}{}^\mathsf{T}y^* = \hat{c} \Leftrightarrow A_\mathcal{B}{}^\mathsf{T}\big(\hat{y} + \frac{y^*}{\Delta_D}\big) = c,$$

which shows the one-to-one correspondence of basic solutions, point 5. □

To our knowledge, no previous articles have described such an iterative refinement algorithm for linear programming. Nevertheless, we believe that some aspects of our approach might have been used in software packages, although most likely not with extended precision or rational arithmetic. We are at least aware that some interior point solvers have experimented with the idea of replacing the objective function of an LP by its reduced cost vector and resolving the problem to improve some of its numerical properties – this would correspond to performing a single dual refinement step.

3.2 Algorithmic details

Now, a few explanatory comments to clarify Algorithm 2. Recall that only the LP solver uses floating-point computation, while the other operations are performed in exact rational arithmetic. The values Δ_P, Δ_D are the, possibly distinct, primal and dual scaling factors; they are used to amplify the error in the right-hand side and variable bounds, and the reduced costs, respectively. They are computed such that the maximum absolute value in the transformed right-hand side, variable bounds, and objective function becomes 1. Because of the limited precision of the corrections returned by the floating-point solver, we additionally limit the increase of the scaling factors at each iteration by some factor α. In our implementation, we chose $\alpha = 10^{12}$, so that $1/\alpha$ is slightly below the numerical tolerance of the LP solver, 10^{-9}.

As stated, Algorithm 2 assumes that the underlying LP solver returns a basic solution in lines 5 and 21. First, this is used to compute the violation of the reduced costs in line 14. If basic information is not available, the test $i \in \mathcal{B}$ can be replaced by checking whether variable x_i is – up to some tolerance – tight at its bound. Second, from point 5 of Proposition 3.1 we know that the corrected solution corresponds to the basis returned for the transformed problem. However, since the floating-point LP solver operates on the approximate problem, the nonbasic variables x_i^{k+1}, $i \notin \mathcal{B}$, may not be at their bound exactly. This is enforced in line 27 to ensure that $\mathcal{B}$ remains a valid starting basis for the transformed problem in the next iteration. If the LP solver does not work with basic solutions, this step can be left out.

Third, since the LPs solved at each iteration are very similar, an algorithm capable of warm starting, such as the simplex method, has a clear advantage. Nevertheless, there may be circumstances where other LP algorithms like interior point methods are desirable and, as explained above, can be applied. If carefully implemented, the iterative refinement algorithm can access the underlying LP solver through an interface and exchange the LP solver whenever beneficial.

Exact computations are used to compute the modified objective function, right-hand side, and variable bounds. The costs of these computations could be reduced by updating

these values from iteration to iteration and by rounding the corrector solution x^*, y^* to have a simple rational representation, similar to the ideas used by Wan [27] for linear system solving. Computing the corrected solution in lines 24 and 25 can be sped up by choosing the scaling factors Δ_P, Δ_D to have a special form, i.e., powers of 2.

Points 1 and 2 of Proposition 3.1 tell us that, assuming we can consistently compute LP solutions that are accurate within an absolute tolerance level of ϵ, then we converge to an arbitrarily precise primal-dual solution. If the violations δ_P and δ_D computed in lines 10 and 14 are always below $1/(\alpha\Delta_P)$ and $1/(\alpha\Delta_D)$, respectively, for $\alpha > 1$, and therefore we increase Δ_P and Δ_D by a factor of α at each iteration, then we obtain a primal-dual solution accurate to an absolute tolerance of $1/\alpha^k$ after k iterations.

An exact rational solution can be computed in one of two ways: first, the exact solution for basis $\mathcal{B}$ can be recomputed exactly, as in Algorithm 1; alternatively, rational reconstruction can be applied directly to the refined solution x^k, y^k.

Although there is no guarantee that a floating-point solver will produce correct solutions that are accurate to within a fixed tolerance level, this is often the case in practice. However, in some cases LPs may be so poorly conditioned that the floating-point LP solver produces meaningless results. In such a case performing extended precision computations within the solver may be necessary. A minor modification could be done to Algorithm 2 to incrementally boost the working precision used for all non-exact computations performed within the for loop when needed, as in Algorithm 1.

3.3 Infeasibility and unboundedness

So far we have described how to compute extended precision solutions for LPs which are indeed primal and dual feasible. Otherwise, our task is to construct an extended precision certificate of the LP's infeasibility or unboundedness.

An LP of form (1) is infeasible if and only if there exists $y \in \mathbb{R}^m$ with $A^\mathsf{T} y \leqslant 0$ and $b^\mathsf{T} y > 0$, a so-called *Farkas* proof. Such a certificate can be detected algorithmically already while trying to solve the LP. More analytically, it is given as dual solution to the slightly modified LP

$$\min\{-\tau : Ax - b\tau = 0, x, \tau \geqslant 0\}, \tag{5}$$

which is trivially feasible. If (and only if) (1) is infeasible then (5) has a finite optimum and we can directly apply Algorithm 2 to compute a certificate of high precision.

A certificate of primal unboundedness consists of a primal feasible point and an unbounded ray $x \in \mathbb{R}^n$ with $Ax = 0$, $x \geqslant 0$, and $c^\mathsf{T} x < 0$. For the former, we may simply apply Algorithm 2 to LP (1) with zero objective function. For the latter, we can compute an extended precision solution to the auxiliary LP

$$\min\{0 : Ax = 0, c^\mathsf{T} x = -1, x \geqslant 0\}, \tag{6}$$

which is primal and dual feasible if and only if (1) is dual infeasible, a consequence of (1) being unbounded.

Now, typically the status of an LP is not known a priori and one starts by trying to solve the original problem. If this is detected as infeasible or unbounded, we can turn to the auxiliary LPs (5) or (6), respectively. However, since floating-point LP solvers relax primal and dual feasibility by numerical tolerances, they may return with a solution claimed optimal, although the LP is infeasible or unbounded. The following shows that nevertheless we may continue by constructing and solving our transformed problem, since certificates of infeasibility or unboundedness remain valid under the transformation.

PROPOSITION 3.2. *Let conditions be given as in Proposition 3.1, then the following hold:*

1. *P is primal unbounded if and only if $\hat{P}$ is, and x^* is an unbounded ray for P if and only if it is an unbounded ray for $\hat{P}$.*
2. *P is primal infeasible if and only if $\hat{P}$ is, and y^* is a Farkas proof for P if and only if it is a Farkas proof for $\hat{P}$.*

PROOF. A primal ray $x^* \geqslant 0$, $Ax^* = 0$, is unbounded for $\hat{P}$ if and only if

$$\Delta_P(c - A^\mathsf{T}\hat{y})^\mathsf{T} x^* < 0 \Leftrightarrow c^\mathsf{T} x^* - \hat{y}^\mathsf{T} A x^* < 0 \Leftrightarrow c^\mathsf{T} x^* < 0,$$

i.e., if it is unbounded for P.

Concerning infeasiblity, note that the Farkas proof for an LP with nonzero lower bounds on the variables, $x \geqslant \ell$, is a dual ray y with $A^\mathsf{T} y \leqslant 0$ and $(b - A\ell)^\mathsf{T} y > 0$. Then y^*, $A^\mathsf{T} y^* \leqslant 0$, is a valid Farkas proof for $\hat{P}$ if and only if

$$\big(\underbrace{\hat{b} - A(-\Delta_p \hat{x})}_{\Delta_P(b - A\hat{x} + A\hat{x})}\big)^\mathsf{T} y^* > 0 \Leftrightarrow b^\mathsf{T} y^* > 0,$$

i.e., if it is a valid Farkas proof for P. □

4. COMPUTATIONAL STUDY

In this section we describe an implementation of Algorithm 2 and evaluate its effectiveness in practice. Experiments were run on a computer with 48 GB RAM and two Intel(R) Xeon(R) X5672 CPUs, each with four 3.2 GHz cores.

4.1 Test environment and test set

We have implemented Algorithm 2 within the SoPlex LP solver, Version 1.6 [25, 29]. SoPlex is an academically developed simplex-based LP solver that is maintained at the Zuse Institute Berlin and is freely available for academic use, including the source code. The rational computations are performed using the GMP arithmetic library [12], Version 4.3.1. By default we are using tolerances $\epsilon_P = \epsilon_D = 0$, scaling limit $\alpha = 10^{12}$, and $k_{\max} = 5$, meaning we perform at most five refinement iterations.

When the transformed problem is solved, on line 20, in addition to warm starting from the previous basis $\mathcal{B}$, we also reuse and keep updating the LU factorization of the basis matrix $A_\mathcal{B}$. This avoids performing a full refactorization of the basis matrix, which SoPlex by default performs only every 200 simplex iterations. LP preprocessing and scaling are not applied. Our current implementation does not include the extensions for infeasible and unbounded LPs.

Henceforth we will use SoPlex$^+$ to denote SoPlex with iterative refinement as described above and SoPlex^0 to specify the standard version of SoPlex without iterative refinement, i.e., using $k_{\max} = 0$.

We now describe our testing framework. In order to observe the effectiveness of Algorithm 2 for computing extended precision solutions we apply SoPlex^0 and SoPlex$^+$ and compare the solution times and solution accuracy. To evaluate the quality of the LP bases identified, we perform

two checks: first, we compute the corresponding primal-dual solution exactly to see if the basis is optimal; second, we use each basis to warm start QSoPT_EX and measure the solution time. This second measurement allows us to observe how helpful iterative refinement can be when embedded within an exact LP solver. Although some information is lost by doing this (in particular, the extended precision solution x^k, y^k constructed in Algorithm 2), it allows for a direct comparison with previous methods.

For calling SoPlex^0, SoPlex$^+$, and QSoPT_EX and for passing the basis from SoPlex to QSoPT_EX we use the exact MIP solving framework described in [7]. All test instances were converted exactly into the form $\min\{c^T x : Ax = b, \ell \leqslant x \leqslant u\}$ by explicitly adding slack variables as described in Section 2.1.

To obtain a meaningful picture of the performance of this algorithm we will present results on two test sets. First, we show the performance on the NETLIB library [21], a standard collection of LP instances, which are known to not be especially numerically difficult. Although most of these LPs were found to be ill-posed in [22], we still consider them to be numerically easy from a practical point of view because floating-point LP solvers often find optimal bases.

For our second test set we collected a pool of numerically troublesome instances with finite optimum. A total of over 900 instances from the NETLIB LP test set [21], Mittelmann's LP test set [20], Mészáros's LP test set (new, miscellaneous, problematic and stochastic test sets) [19], and LP relaxations of all instances in the MIPLIB 2010 test library [18] were taken as a starting point. From this collection we selected the instances for which SoPlex^0 did not identify an optimal basis. We eliminated some instances with excessive running times (over 24 hours) and one instance, `iprob`, that was too poorly conditioned for the double-precision LP solver to handle. Finally, after removing some instances of the overrepresented classes `delf0**`, `large0**`, `small0**`, we were left with 74 instances.

At last, we need to comment on the tolerances used by floating-point LP solvers to measure primal and dual feasibility. Generally, using a stricter tolerance will more frequently find an optimal LP basis, while too strict of a tolerance can lead to a numerical breakdown. By default, SoPlex uses an absolute feasibility tolerance of 10^{-6}. In our experiments, we have tightened it to 10^{-9}, which we observed to find more optimal bases than the default. Further tightening this tolerance to 10^{-12} enabled SoPlex^0 to identify optimal bases for some additional instances but yielded numerical troubles and incorrect claims of infeasibility or unboundedness on others. We note that the analogous tolerance in the commercial LP solver CPLEX is adjustable by the user to no smaller than 10^{-9}.

4.2 Computational results

We first present results on the NETLIB LP instances. Table 1 lists aggregate information for SoPlex^0 and SoPlex$^+$: the number of LP iterations used "LP Iters", the LP solution time "LP Time [sec]" (including iterative refinement, if used), the time required by QSoPT_EX to solve the LP exactly after being warm started from the final LP basis of SoPlex^0 or SoPlex$^+$ "EXLP Time [sec]", and finally the number of instances where an optimal basis was found. Average times and LP iterations are reported as geometric means because solution times and LP iterations vary widely among the instances. Times were rounded up to 0.1 second if smaller.

Table 1: Results on NETLIB LP test set.

	SoPlex^0	SoPlex$^+$
LP Iters	773	775
LP Time [sec]	0.21	0.34
EXLP Time [sec]	0.21	0.19
Optimal basis found	89/92	92/92

The main observations we make from Table 1 are that performing some iterative refinement on instances that do not have numerical problems does not introduce a significant overhead, neither in solution time nor in the number of simplex pivots. Once an optimal basis is found, the iterative refinement rounds simply refine the accuracy of the basic solution without performing additional pivots. Note that SoPlex$^+$ found all optimal bases whereas SoPlex^0 fails on `dfl001`, `etamacro`, and `pilot87`, which are also included in our troublesome test set. When Koch [17] identified all optimal NETLIB bases with SoPlex, 128-bit arithmetic was used on some instances.

Table 2 presents computational results on the numerically troublesome instances, comparing SoPlex^0 with SoPlex$^+$. For these two methods we list the total number of simplex pivots used "LP Iters", the LP solution time "LP Time [sec]" (including iterative refinement, if used), and the time required by QSoPT_EX to solve the LP exactly when warm started from the final basis returned by SoPlex^0 or SoPlex$^+$ "EXLP Time [sec]". For both solvers we also list the maximum error "Violation" of the solution returned (violation of constraints, bounds, or reduced costs), computed exactly.

Note that although we always perform five rounds of iterative refinement, SoPlex$^+$ might already arrive at the final basis in an earlier round. The remaining rounds then only refine the numerical solution at this basis without further simplex pivots. Column "Rnds to Bas" reports the round in which SoPlex$^+$ identified the basis that it returned. The final row states geometric mean values; here, times were rounded up to 0.1 second if smaller.

From Table 2 we see that SoPlex$^+$ successfully computed high precision solutions without experiencing a large overhead in time or number of LP iterations when compared to SoPlex^0. Better yet, SoPlex$^+$ returned an optimal basis for every instance; in all but three cases this basis was found within the first or second round of refinement.

Finally, to evaluate the improvement that iterative refinement provides to exact LP solving, we consider the time used by SoPlex plus the time of QSoPT_EX after warm starting for both SoPlex^0 and SoPlex$^+$. Considering the arithmetic mean of these ratios, iterative refinement is 9.76 times faster, with ratios on individual instances as high as 239 (`mod2`). (Alternatively, computed as a ratio of the geometric means in the last column of Table 2, the speedup factor is 5.45.) Apart from the tiny instance `gams30a`, the only instance with a slowdown is `fome13`, which is explained by an anomalous situation in QSoPT_EX: although SoPlex$^+$ computed an optimal basis, due to numerical errors the first double-precision solve (on line 4 of Algorithm 1) failed to recognize the dual feasibility of the basis and then proceeded to solve the instance from scratch, resulting in the long running time.

Table 2: Results on troublesome test instances.

	SoPlex^0 (SoPlex without iterative refinement)				SoPlex$^+$ (SoPlex with iterative refinement)				
Name	LP Iters	LP Time [sec]	EXLP Time [sec]	Violation	LP Iters	LP Time [sec]	EXLP Time [sec]	Violation	Rnds to Bas
atm20-100	4243	0.29	12.15	2.00373e-11	4364	1.16	0.17	1.85688e-73	1
delf000	2300	0.26	3.39	5.02016e-13	2432	0.54	0.56	8.39421e-90	1
delf001	2409	0.27	4.16	4.78653e-13	2545	0.66	0.48	4.83543e-89	1
delf002	2491	0.40	4.20	6.17014e-13	2624	0.71	0.63	2.64309e-87	1
delf003	2894	0.43	4.06	9.57791e-13	3031	0.79	0.98	1.16510e-88	1
delf004	2969	0.34	4.55	8.77790e-13	3111	0.75	1.18	6.13890e-88	1
delf005	3481	0.42	4.94	1.76242e-12	3618	0.88	1.22	5.46639e-87	1
delf006	3191	0.41	7.61	2.44232e-12	3328	0.77	1.82	1.91097e-88	1
delf007	3198	0.46	7.00	1.20085e-12	3376	0.80	1.25	3.82540e-87	1
delf008	3247	0.49	7.20	2.58426e-12	3417	0.85	1.40	3.21860e-82	2
delf009	3531	0.56	8.96	1.66767e-12	3701	1.02	1.96	9.97465e-88	1
delf010	3182	0.53	7.52	2.61187e-12	3354	0.95	1.31	6.57537e-83	2
delf011	3151	0.36	6.99	1.80400e-12	3321	0.84	0.69	9.56086e-88	1
delf012	3627	0.51	6.10	1.66206e-12	3796	0.92	0.99	9.65252e-88	1
delf013	3452	0.44	6.20	1.15851e-11	3625	0.87	1.10	1.96875e-88	1
delf014	3164	0.36	7.25	2.04394e-12	3305	0.78	0.84	7.13000e-89	1
delf015	3666	0.52	7.38	2.84292e-12	3807	0.94	1.01	1.43751e-88	1
delf017	3265	0.38	6.30	7.29140e-13	3406	0.86	0.86	2.85603e-88	1
delf018	3369	0.40	5.87	7.28550e-13	3512	0.92	0.54	1.30510e-87	1
delf019	3165	0.39	4.78	6.49831e-13	3308	0.72	0.40	1.39656e-88	1
delf020	3302	0.47	7.04	3.09865e-12	3492	0.85	0.53	3.55083e-81	2
dfl001	23175	15.31	4127.79	4.33902e-10	23177	16.27	1.04	2.59988e-68	1
etamacro	936	0.08	0.40	9.99997e-10	937	0.21	0.04	3.63132e-76	1
fome11	42323	44.57	1884.00	2.83435e-10	42328	47.34	1.91	1.27600e-68	1
fome12	92070	149.89	7192.94	7.41202e-10	92267	155.80	3.55	2.76417e-63	1
fome13	183383	452.70	9135.32	9.07219e-10	184744	471.67	10297.09	3.41988e-60	1
gams30a	487	0.02	0.08	1.00000e-09	500	0.10	0.02	1.70479e-87	1
gen1	3628	8.25	1406.45	8.43854e-10	3978	10.23	311.25	1.67594e-66	1
gen	3628	8.32	1417.51	8.43854e-10	3978	10.25	311.45	1.67594e-66	1
large000	3828	0.33	9.09	6.20093e-13	4007	0.84	0.68	2.28257e-90	1
large001	6468	1.05	6.91	4.03544e-12	6647	1.66	1.00	6.66322e-81	1
large002	4627	0.60	13.11	5.60934e-12	4952	1.26	1.96	1.07875e-84	2
large003	4928	0.69	10.31	1.58380e-12	5102	1.28	1.24	9.47192e-89	1
large004	5134	0.78	11.14	3.01933e-11	5313	1.37	2.14	2.23100e-87	1
large005	4931	0.60	10.12	8.00677e-13	5104	1.15	1.08	1.23062e-88	1
large006	4840	0.59	11.12	6.85503e-13	5059	1.15	1.43	2.03351e-86	1
large007	5056	0.67	9.50	1.32355e-12	5277	1.26	1.36	2.53356e-87	1
large008	5224	0.76	10.15	9.29166e-13	5445	1.33	1.48	1.61060e-86	1
large009	5278	0.75	11.55	1.20416e-12	5498	1.34	1.59	5.94184e-88	1
large010	4709	0.59	11.12	4.01766e-11	4929	1.20	1.48	1.33615e-80	2
large011	5176	0.71	11.50	8.87912e-13	5396	1.29	1.71	8.51599e-88	1
large012	4763	0.60	11.26	6.93811e-12	4983	1.21	1.40	7.10682e-86	1
large013	5254	0.75	10.40	1.47538e-12	5475	1.33	1.14	1.52980e-87	1
large014	5091	0.66	10.32	6.81819e-13	5314	1.23	1.16	9.11155e-88	1
large015	5111	0.63	10.26	1.39860e-12	5345	1.22	1.22	7.14458e-86	1
large016	4878	0.56	10.74	2.20265e-12	5113	1.15	1.18	3.45056e-87	1
large017	4705	0.50	9.63	5.18197e-12	4884	1.07	1.06	3.74349e-88	1
large018	4930	0.56	9.83	4.85757e-12	5109	1.14	0.75	1.93616e-88	1
large019	4833	0.50	9.47	5.88247e-13	5012	1.05	0.65	6.66263e-90	1
large020	5263	0.92	14.05	9.76857e-13	5501	1.24	0.79	9.86219e-88	1
mod2	69220	152.20	13428.22	3.47427e-10	69330	159.34	6.05	1.43436e-46	3
momentum3	79570	305.97	15935.44	8.61580e-08	79574	388.06	2081.02	1.21275e-54	1
msc98-ip	126430	107.28	937.60	1.31193e-07	126553	109.15	0.42	5.58897e-71	1
noswot	221	0.01	0.06	2.89565e-10	229	0.05	0.01	9.10199e-88	1
ns1904248	167345	784.86	3796.06	9.60730e-10	167785	728.52	5.90	6.13848e-79	2
pilot87	11381	10.95	3635.76	3.52688e-10	11383	14.52	1817.25	7.33788e-60	1
rail01	142104	406.35	5517.79	5.31826e-14	142181	414.17	2.28	2.52325e-74	1
ran14x18-disj-8	621	0.12	12.26	1.96998e-12	622	0.66	4.11	1.27150e-68	1
seymour-disj-10	8018	2.93	106.76	6.20706e-10	8020	4.55	6.89	5.60644e-73	1
small000	705	0.03	0.48	6.66658e-12	738	0.25	0.07	4.27459e-92	1
small001	795	0.03	0.52	4.09592e-13	824	0.25	0.08	9.78188e-89	1
small002	790	0.02	0.64	5.25500e-13	821	0.26	0.16	2.20124e-86	1
small003	794	0.05	0.55	6.66705e-12	828	0.26	0.08	3.70605e-90	1
small004	789	0.04	0.53	6.66654e-13	823	0.25	0.07	8.97819e-91	1
small005	764	0.05	0.55	7.46174e-13	798	0.27	0.08	8.22301e-91	1
small006	764	0.07	0.52	6.66733e-13	807	0.27	0.07	6.28624e-91	1
small007	678	0.05	0.61	6.66733e-13	740	0.28	0.06	3.17979e-84	2
small008	672	0.04	0.58	4.37694e-13	732	0.26	0.08	4.71453e-85	2
small009	628	0.04	0.76	1.16554e-12	688	0.25	0.07	1.86784e-89	1
small010	606	0.05	0.74	4.66169e-13	646	0.12	0.06	1.10441e-89	1
stat96v5	4192	7.16	3692.31	5.54481e-10	4527	14.60	1653.70	1.10266e-59	1
unitcal_7	57353	25.50	38.07	5.19549e-11	57363	29.58	0.99	1.95114e-74	4
watson_1	224894	847.72	3555.79	9.96540e-10	225192	883.64	17.83	1.20810e-75	1
world	84623	209.02	8714.77	9.64974e-10	84665	221.66	5.84	4.61431e-31	4
Instances (74)	4680	1.11	17.43		4863	2.00	1.40		

5. CONCLUSION

In summary, we have described an iterative refinement algorithm for linear programming and demonstrated it to be efficient in practice for computing extended precision solutions on a wide range of LPs coming from industrial applications. Additionally, when used as a subroutine for solving the LPs exactly, iterative refinement is considerably faster than using extended precision pivoting.

Future research directions could include a tighter integration of incremental precision boosting, rational reconstruction, and iterative refinement. Such an integration could both improve the speed and applicability of these techniques, in particular on instances too poorly conditioned for the double-precision LP subroutines.

6. ACKNOWLEDGMENTS

Kati Wolter was supported by the DFG Priority Program 1307 "Algorithm Engineering". Many thanks to Timo Berthold for his valuable comments.

7. REFERENCES

[1] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza. Exact solutions to linear programming problems. *Oper. Res. Lett.*, 35(6):693–699, 2007.

[2] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza. QSopt_ex, 2007. `http://www.dii.uchile.cl/~daespino/ESolver_doc/`.

[3] C. Buchheim, M. Chimani, D. Ebner, C. Gutwenger, M. Jünger, G. Klau, P. Mutzel, and R. Weiskircher. A branch-and-cut approach to the crossing number problem. *Discrete Optimization*, 5(2):373 – 388, 2008.

[4] D. Bulutoglu and D. Kaziska. Improved WLP and GWP lower bounds based on exact integer programming. *J. of Stat. Plan. and Inference*, 140(5):1154 – 1161, 2010.

[5] B. A. Burton and M. Ozlen. Computing the crosscap number of a knot using integer programming and normal surfaces. *Preprint available on arXiv.org*, 2011.

[6] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.

[7] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. An exact rational mixed-integer programming solver. In O. Günlük and G. Woeginger, editors, *Integer Programming and Combinatoral Optimization*, volume 6655 of *LNCS*, pages 104–116. Springer Berlin / Heidelberg, 2011.

[8] W. Cook and D. E. Steffy. Solving very sparse rational systems of equations. *ACM Trans. on Math. Software*, 37(4), 2011.

[9] F. M. de Oliveira Filho and F. Vallentin. Fourier analysis, linear programming, and densities of distance avoiding sets in $\mathbb{R}^n$. *J. Eur. Math. Soc.*, 12(6):1417–1428, 2010.

[10] M. Dhiflaoui, S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schömer, R. Schulte, and D. Weber. Certifying and repairing solutions to large LPs: How good are LP-solvers? In *Proceedings of the 14th annual symposium on Discrete algorithms*, SODA '03, pages 255–256, Philadelphia, PA, USA, 2003. SIAM.

[11] Daniel G. Espinoza. *On Linear Programming, Integer Programming and Cutting Planes*. Ph.D. thesis, Georgia Institute of Technology, 2006.

[12] GMP, GNU Multiple Precision Arithmetic Library Version 4.3.1. `http://gmplib.org/`.

[13] G. Golub and C. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, USA, 1983.

[14] T. C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005.

[15] S. Held, W. Cook, and E. Sewell. Safe lower bounds for graph coloring. In O. Günlük and G. Woeginger, editors, *Integer Programming and Combinatoral Optimization*, volume 6655 of *LNCS*, pages 261–273. Springer Berlin / Heidelberg, 2011.

[16] I. Hicks and N. McMurray. The branchwidth of graphs and their cycle matroids. *Journal of Combinatorial Theory Series B*, 97(5):681–692, 2007.

[17] T. Koch. The final NETLIB-LP results. *Operations Research Letters*, 32(2):138–142, 2004.

[18] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Math. Program. Comp.*, 3(2):103–163, 2011.

[19] Cs. Mészáros. LP test set. `http://www.sztaki.hu/~meszaros/public_ftp/lptestset/`, 2006.

[20] H. D. Mittelmann. Benchmarks for Optimization Software. `http://plato.asu.edu/bench.html`, 2010.

[21] NETLIB. LP library. `http://www.netlib.org/lp/`.

[22] F. Ordóñez and R. Freund. Computational experience and the explanatory value of condition measures for linear optimization. *SIAM J. on Optimization*, 14(2):307–333, 2003.

[23] V. Y. Pan. Nearly optimal solution of rational linear systems of equations with symbolic lifting and numerical initialization. *Computers & Mathematics with Applications*, 62(4):1685–1706, 2011.

[24] B. D. Saunders, D. H. Wood, and B. S. Youse. Numeric-symbolic exact rational linear system solver. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation*, ISSAC '11, pages 305–312, New York, NY, USA, 2011. ACM.

[25] SoPlex Version 1.6. `http://soplex.zib.de/`.

[26] S. Ursic and C. Patarra. Exact solution of systems of linear equations with iterative methods. *SIAM Journal on Matrix Analysis and Applications*, 4(1):111–115, 1983.

[27] Z. Wan. An algorithm to solve integer linear systems exactly using numerical methods. *J. of Symbolic Computation*, 41(6):621–632, 2006.

[28] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice Hall, Englewood Cliffs, NJ, 1963.

[29] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.

[30] Chee K. Yap. Robust geometric computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of discrete and computational geometry*, pages 653–668. CRC Press, Inc., Boca Raton, FL, USA, 1997.