INFORMS Journal on Computing
Vol. 28, No. 3, Summer 2016, pp. 1–16
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2016.0692
© 2016 INFORMS

Iterative Refinement for Linear Programming

Ambros M. Gleixner
Zuse Institute Berlin, Department Optimization, 14195 Berlin, Germany, gleixner@zib.de

Daniel E. Steffy
Mathematics and Statistics, Oakland University, Rochester, Michigan 40309, steffy@oakland.edu

Kati Wolter
MOSEK ApS, 2100 Copenhagen, Denmark, kati.wolter@mosek.com

We describe an iterative refinement procedure for computing extended-precision or exact solutions to linear programming (LP) problems. 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, and 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 show that this algorithm is effective in practice for computing extended-precision solutions and that it leads to direct improvement of the best known methods for solving LPs exactly over the rational numbers. Our implementation is publically available as an extension of the academic LP solver SoPlex.

Keywords: linear programming; exact computation; iterative refinement; rational arithmetic; high-precision optimization; accurate solutions
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received May 2015; revised October 2015; accepted December 2015.

1. Introduction

Most fast linear programming solvers available today for solving linear programs (LPs) use floating-point arithmetic, which can lead to numerical errors. Although such implementations are effective at computing approximate solutions for a wide range of instances, there are situations when they give unreliable results, 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 (Applegate et al. 2007a, Cook et al. 2011).

Some recent articles that have used the exact solution of linear or integer programming instances to establish theoretical results, or to solve instances from numerically demanding applications, include Buchheim et al. (2008), Bulutoglu and Kaziska (2010), Burton and Ozlen (2012), Cohn et al. (2011), de Oliveira Filho and Vallentin (2010), Hales (2005), Held et al. (2012), Hicks and McMurray (2007), Lerman et al. (2012), and Chindelevitch et al. (2014).

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.

Our main contribution is a new algorithm based on iterative refinement that builds extended-precision LP solutions using an approximate LP solver as a subroutine. As a byproduct, this algorithm helps to accelerate the state of the art for solving LPs exactly over the rational numbers.

This article is a significant extension of Gleixner et al. (2012). It is organized as follows. In Section 2 we give an overview of previous methods and introduce notation. Section 3 introduces iterative refinement for primal and dual feasible LPs and discusses convergence and computational efficiency of the algorithm. In Section 4 we address the case of infeasible or unbounded LPs and give an integrated algorithm to handle LPs with a priori unknown status. Section 5 describes our implementation within the academic LP solver SoPlex and investigates the performance of the algorithm over a test set of 1,202 publicly available benchmark instances. We give concluding remarks in Section 6. Proofs, generalizations, and extended computational results can be found in an online supplement (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2016.0692).

2. Previous Work

We assume that the reader is familiar with the concepts of linear programming and refer to Dantzig (1963), Chvátal (1983), or Schrijver (1986) for details. For clarity of presentation, we assume that a LP is given in standard form

$$\min\{c^{\mathsf{T}}x \mid Ax = b,\, x \geq l\}, \tag{1}$$

where $c \in \mathbb{Q}^n$ is the objective function vector, $l \in \mathbb{Q}^n$ is the vector of lower bounds, $A \in \mathbb{Q}^{m\times n}$ is the constraint matrix, and $b \in \mathbb{Q}^m$ is the right-hand side. Extensions to LPs in general form are discussed in Section B of the online supplement. Without loss of generality, we assume A has full row rank and $n \geq m$. We assume rational input data. The *dual LP* of (1) reads

$$\max\{b^{\mathsf{T}}y + l^{\mathsf{T}}z \mid A^{\mathsf{T}}y + z = c,\, z \geq 0\}, \tag{2}$$

where $z = c - A^{\mathsf{T}}y$ is the vector of *dual slacks*. Because these are uniquely determined by the vector of dual multipliers y, we sometimes speak of a dual solution y when we actually mean a solution (y, z).

Most LP algorithms produce pairs of primal-dual solutions x, y. They are optimal if and only if x is primal feasible, y is dual feasible, and complementary slackness is satisfied; i.e., their *duality gap* is zero, defined as $\gamma(x, y) = (x - l)^{\mathsf{T}}(c - A^{\mathsf{T}}y)$. They are called *basic* if there exists a subset of column indices $\mathcal{B} \subseteq \{1, \ldots, n\}$, $|\mathcal{B}| = m$, called a *basis* such that the *basis matrix* $A_{\cdot\mathcal{B}}$ is regular and x and y are uniquely determined by the linear systems $A_{\cdot\mathcal{B}}x_{\mathcal{B}} = b - \sum_{i\notin\mathcal{B}} A_{\cdot i}l_i$ and $y^{\mathsf{T}}A_{\cdot\mathcal{B}} = c_{\mathcal{B}}^{\mathsf{T}}$.[1] Geometrically, the primal feasible basic solutions are the vertices of the feasible region.

2.1. Exact Methods for Linear Programming over the Rational Numbers

There is a trivial method of solving LPs with rational input data exactly, which is to apply a simplex algorithm and perform all computations in (exact) rational arithmetic.[2] Because of the high computational cost of rational arithmetic, this approach becomes prohibitively slow for large instances. More precisely, as Espinoza (2006) demonstrates computationally, the running time of the naïve approach is not so much correlated with the size of an LP, but with the encoding length of the basic solutions traversed by the simplex algorithm. Notable improvements of this approach are Edmonds' Q-pivoting (see Edmonds 1994, Edmonds and Maurras 1997, Azulay and Pique 1998) and the mixed-precision simplex algorithm of Gärtner (1999).

Recent research efforts exploit the basic information provided by the simplex algorithm. If a candidate for an optimal basis is identified, then the corresponding primal-dual solution can be computed exactly and checked. If it is primal and dual feasible, then it is optimal, because by construction basic solutions satisfy the complementary slackness conditions. It has been observed by Dhiflaoui et al. (2003) and Koch (2004) that LP bases returned by floating-point solvers are often optimal. Applegate et al. (2007b) developed an exact rational LP solver, QSopt_ex, that exploits this behavior to achieve fast average computation times. 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.

Algorithm 1 (Incremental precision boosting for a primal and dual feasible LP)

```
  in: $\min\{c^{\mathsf{T}}x \mid Ax = b, x \geq l\}$ with $A \in \mathbb{Q}^{m\times n}$,
      $b \in \mathbb{Q}^m, c, l \in \mathbb{Q}^n$
  out: primal-dual solution $x^* \in \mathbb{Q}^n$, $y^* \in \mathbb{Q}^m$, basis
      $\mathcal{B} \subseteq \{1, \ldots, n\}$
1 begin
2   for $p \leftarrow$ double, 128, 256, ..., rational do
3     get $\bar{A}, \bar{b}, \bar{c}, \bar{l} \approx A, b, c, l$ in precision $p$
4     solve $\min\{\bar{c}^{\mathsf{T}}x \mid \bar{A}x = \bar{b}, x \geq \bar{l}\}$ in precision $p$
5     get basis $\mathcal{B}$ returned as optimal
6     solve $A_{\cdot\mathcal{B}}x^*_{\mathcal{B}} = b - \sum_{i\notin\mathcal{B}} A_{\cdot i}l_i$ and $y^{*\mathsf{T}}A_{\cdot\mathcal{B}} = c^{\mathsf{T}}_{\mathcal{B}}$
        in rational arithmetic
7     if $x^*_{\mathcal{B}} \geq l_{\mathcal{B}}$ and $A^{\mathsf{T}}y^* \leq c$ then
8       foreach $i \notin \mathcal{B}$ do $x^*_i \leftarrow l_i$
9       return $x^*, y^*, \mathcal{B}$
```

QSopt_ex is often effective at finding exact solutions quickly, especially when the double-precision LP subroutines are able to find an optimal LP basis. However, in cases when 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.

2.2. Iterative Refinement for Linear Systems of Equations

Iterative refinement is a common technique to improve numerical accuracy when solving linear systems of equations, going back to Wilkinson (1963). Given a system of linear equations $Mx = r$, $M \in \mathbb{Q}^{n\times n}$, $r \in \mathbb{Q}^n$, a sequence of increasingly accurate solutions $\{x_1, x_2, \ldots\}$ is constructed by first computing an approximate solution x_1 with $Mx_1 \approx r$. Then for $k \geq 2$, a refined solution $x_k \leftarrow x_{k-1} + \hat{x}$ is computed, where $\hat{x}$ satisfies $M\hat{x} \approx \hat{r}$ and is a correction of the error $\hat{r} = r - Mx_{k-1}$ 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

[1] Here we use the notation $A_{\mathcal{I},\mathcal{J}}$ for the submatrix of A with rows and columns restricted to index sets $\mathcal{I} \subseteq \{1, \ldots, m\}$ and $\mathcal{J} \subseteq \{1, \ldots, n\}$, similarly for vectors. We use "$\cdot$" to abbreviate the set of all columns or rows.

[2] Implementations of this approach can be found, e.g., in the packages for discrete computational geometry cdd+ (see Fukuda and Prodon 1996 and lrs by Avis and Fukuda 1992).

level of precision, or in *mixed precision*, where the computation of the residual errors $\hat{r}$ and the addition of the correction are computed with a higher level of precision than the system solves (see, e.g., Golub and van Loan 1983 for more details).

2.3. Moving from Approximate to Exact Solutions

Iterative refinement can also be used as a subroutine for computing exact solutions to rational systems of linear equations. After a sufficiently accurate solution has been constructed, the exact rational solution vector can be recovered using continued fraction approximations, as computed by the extended Euclidean algorithm. This idea was first described by Ursic and Patarra (1983) and improved upon by Wan (2006), Pan (2011), and Saunders et al. (2011). Cook and Steffy (2011) compared this to other strategies for computing the exact rational solutions in line 6 of Algorithm 1. In many cases it was the fastest method.

The idea of "rounding" approximate solutions to exact rational solutions has been applied in more theoretical contexts as well. Grötschel et al. (1988) give polynomial-time LP algorithms based on the method of Khachiyan (1979), each iteration of which produces a smaller and smaller ellipsoid enclosing an optimal solution. Using this ability to find a tight enclosure around an optimal solution, techniques of lattice basis reduction can be applied to recover an exact rational solution. For further discussion of these techniques, referred to as *Diophantine approximation*, see Schrijver (1986), Grötschel et al. (1988), and Von zur Gathen and Gerhard (2003). We also refer the reader to Yap (1997) 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.

2.4. Rigorous Bounds

Finally, we want to mention a different line of research that has focused on computing rigorous bounds on the objective function value of an LP. This can be particularly useful when solving mixed-integer linear programs exactly; see Jansson (2004), Neumaier and Shcherbina (2004), Althaus and Dumitriu (2009), and Steffy and Wolter (2013), for more details. These approaches employ interval arithmetic, see, e.g., Moore et al. (2009), in place of rational arithmetic in parts or throughout their algorithms.

3. Iterative Refinement for Linear Programming

Iterative refinement is already applied by many floating-point LP solvers to improve their numerical robustness when solving linear systems (Maes 2013). We take this idea one step further and show how iterative refinement can be applied to an entire LP.

3.1. The Basic Algorithm

Our method solves a sequence of LPs, each one computing a correction of the previous to build an accurate primal-dual solution. This strategy will simultaneously refine both the primal and dual solutions by adjusting the primal feasible region and the objective function of the LP to be solved. It is based on the following theorem, which formally holds for all positive scaling factors Δ_P, Δ_D. As will become clear, we are interested in the case when $\Delta_P, \Delta_D \gg 1$.

Theorem 1. *Suppose we are given an LP (P) in form $\min\{c^\top x \mid Ax = b, x \geq l\}$; then for $x^* \in \mathbb{R}^n$, $y^* \in \mathbb{R}^m$, and scaling factors $\Delta_P, \Delta_D > 0$, consider the transformed problem*

$$\min\{\Delta_D \hat{c}^\top x \mid Ax = \Delta_P \hat{b}, x \geq \Delta_P \hat{l}\}, \qquad (\hat{P})$$

where $\hat{c} = c - A^\top y^*$, $\hat{b} = b - Ax^*$, and $\hat{l} = l - x^*$. Then for any $\hat{x} \in \mathbb{R}^n$, $\hat{y} \in \mathbb{R}^m$ the following hold:

1. *$\hat{x}$ is primal feasible for $\hat{P}$ within an absolute tolerance $\epsilon_P \geq 0$ if and only if $x^* + (1/\Delta_P)\hat{x}$ is primal feasible for P within ϵ_P/Δ_P.*
2. *$\hat{y}$ is dual feasible for $\hat{P}$ within an absolute tolerance $\epsilon_D \geq 0$ if and only if $y^* + (1/\Delta_D)\hat{y}$ is dual feasible for P within ϵ_D/Δ_D.*
3. *$\hat{x}, \hat{y}$ violate complementary slackness for $\hat{P}$ by $\epsilon_S \geq 0$ if and only if $x^* + (1/\Delta_P)\hat{x}$, $y^* + (1/\Delta_D)\hat{y}$ violate complementary slackness for P by $\epsilon_S/(\Delta_P\Delta_D)$.*
4. *$\hat{x}, \hat{y}$ is an optimal primal-dual solution for $\hat{P}$ if and only if $x^* + (1/\Delta_P)\hat{x}$, $y^* + (1/\Delta_D)\hat{y}$ is optimal for P.*
5. *$\hat{x}, \hat{y}$ is a basic primal-dual solution of $\hat{P}$ associated with basis $\mathcal{B}$ if and only if $x^* + (1/\Delta_P)\hat{x}$, $y^* + (1/\Delta_D)\hat{y}$ is a basic primal-dual solution for P associated with basis $\mathcal{B}$.*

Proof. See Section A.1 of the online supplement. □

Theorem 1 can be viewed in two complementary ways. From a numerical perspective, $(\hat{P})$ is formed by replacing the right-hand side b, the bounds on the variables l, and the objective function vector c by the corresponding residual errors in an approximate solution x^*, y^*. This is similar to an iterative refinement step for linear systems of equations, but additionally the residual errors are magnified by the scaling factors Δ_P and Δ_D. Points 1–3 state the improved accuracy of the corrected solution $x^* + (1/\Delta_P)\hat{x}$, $y^* + (1/\Delta_D)\hat{y}$ if $\hat{x}, \hat{y}$ is an approximate solution to $(\hat{P})$.

Geometrically, $(\hat{P})$ is the result of applying the affine transformation $x \mapsto \Delta_P(x - x^*)$ to the primal and $y \mapsto \Delta_D(y - y^*)$ to the dual solution space of (P). Theorem 1 summarizes the straightforward one-to-one correspondence between solutions of the original problem (P) and the transformed problem $(\hat{P})$. Graphically, the primal transformation zooms in on the reference solution x^*—by first shifting the reference solution x^* to the origin, then scaling the problem by a factor of Δ_P. The dual transformation tilts the objective function to

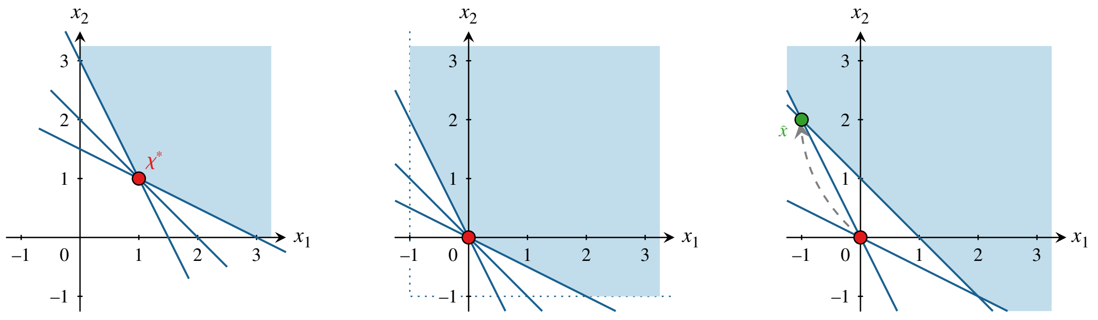

Figure 1 (Color online) Two-Variable Example for Primal LP Refinement

become the vector of reduced cost. This is illustrated by the following examples.

Example 1 (Primal LP Refinement). Consider the LP on two variables

$$\min\{x_1 + x_2 \mid 2x_1 + x_2 \geq 3,\, x_1 + 2x_2 \geq 3,$$
$$x_1 + x_2 \geq 2 + 10^{-6},\, x_1, x_2 \geq 0\},$$

with an approximate solution $x^* = (1, 1)^\mathsf{T}$ as depicted in Figure 1(a). (We use the inequality form without slack variables here for better visualization.) Note that the constraint $x_1 + x_2 \geq 2 + 10^{-6}$ is indistinguishable from $x_1 + x_2 \geq 2$ and the tiny violation of 10^{-6} is invisible on this scale. Shifting the problem such that the reference solution is centered at the origin gives the shifted LP in Figure 1(b). After scaling the primal space by $\Delta_P = 10^6$, we obtain the transformed problem $\min\{x_1 + x_2 \mid 2x_1 + x_2 \geq 0,\, x_1 + 2x_2 \geq 0,\, x_1 + x_2 \geq 1,\, x_1, x_2 \geq -10^6\}$ in Figure 1(c)—$(\hat{P})$ in Theorem 1. Here, the infeasibility of the initial solution is apparent. An LP solver might return the solution $\hat{x} = (-1, 2)^\mathsf{T}$ instead, which corresponds to the corrected, exactly feasibly solution $x^* + (1/\Delta_P)\hat{x} = (1 - 10^{-6}, 1 + 2 \cdot 10^{-6})$ for the original problem.

Example 2 (Dual LP Refinement). Consider the LP on two variables

$$\min\{x_1 + (1 - 10^{-6})x_2 \mid x_1 + x_2 = 2,\, x_1, x_2 \geq 0\},$$

with an approximate solution $x^* = (2, 0)^\mathsf{T}$, as shown in Figure 2(a). Note that any point on the line $x_1 + x_2 = 2$ looks optimal because on this scale the objective function is not distinguishable from $x_1 + x_2$. With dual multiplier $y^* = 1$ the reduced cost vector is $\hat{c} = (0, -10^{-6})^\mathsf{T}$. The solution x^*, y^* satisfies complementary slackness but is dual infeasible and slightly suboptimal. After replacing the objective function with the reduced cost vector and scaling it by $\Delta_D = 10^6$, we obtain the transformed LP with objective function $-x_2$ in Figure 2(b). Now, the initial solution is seen to be clearly suboptimal and any LP solver should return the solution $\hat{x} = (0, 2)^\mathsf{T}$ instead, which—because we did not transform the primal in this example—is already the corrected, now optimal solution for the original LP. Alternatively, we can view this dual refinement as a primal-refinement step on the one-dimensional dual LP $\max\{2y \mid y \leq 1,\, y \leq 1 - 10^{-6}\}$. Shifting by the approximate dual solution $y^* = 1$ and scaling by $\Delta_D = 10^6$ yields the transformed problem $\max\{2y \mid y \leq 0,\, y \leq -1\}$. This gives the dual corrector $\hat{y} = -1$ and the corrected dual solution $y^* + (1/\Delta_D)\hat{y} = 1 - 10^{-6}$.

Applying the refinement step of Theorem 1 iteratively gives the scheme outlined in Algorithm 2. First, the LP is solved approximately, producing an initial primal-dual solution x_k, y_k for $k = 1$. Then, the primal and dual residual errors are computed and used to check whether termination tolerances for primal and dual feasibility and complementary slackness have been reached. If not, then the transformed problem $(\hat{P})$ is set up as in Theorem 1. The scaling factors are chosen as the inverse of the maximum primal and dual violations to normalize the right-hand side, lower bound, and objective function vectors. Additionally, we limit the increase of the scaling factors from round to round by the incremental scaling factor α to ensure that we do not scale by infinity if one of the violations drops to zero. Eventually, the transformed LP is solved approximately to obtain a solution $\hat{x}, \hat{y}$, which is used to refine the accuracy of the candidate solution x_k, y_k.

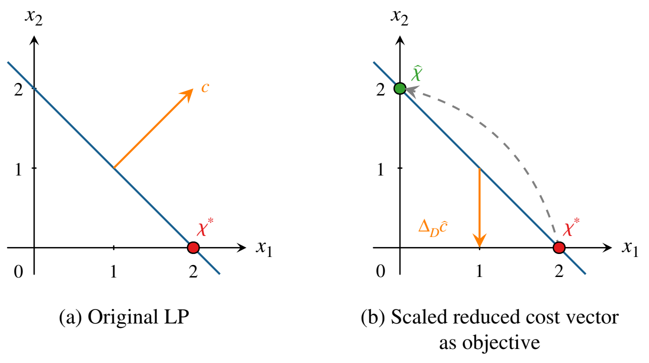

Figure 2 (Color online) Two-Variable Example for Dual LP Refinement

Algorithm 2 (Iterative refinement for a primal and dual feasible LP)

```
in: $\min\{c^\mathsf{T}x \mid Ax = b, x \geq l\}$ with $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c, l \in \mathbb{Q}^n$, incremental scaling limit $\alpha > 1$, termination
    tolerances $\epsilon_P, \epsilon_D, \epsilon_S > 0$
out: primal–dual solution $x^* \in \mathbb{Q}^n$, $y^* \in \mathbb{Q}^m$
1  begin
2      $\Delta_{P,1} \leftarrow 1$, $\Delta_{D,1} \leftarrow 1$                                   /* initial solve */
3      get $\bar{A}, \bar{b}, \bar{l}, \bar{c} \approx A, b, l, c$ in precision of the LP solver
4      solve $\min\{\bar{c}^\mathsf{T}x \mid \bar{A}x = \bar{b}, x \geq \bar{l}\}$ approximately
5      $x_1, y_1 \leftarrow$ approximate primal–dual solution returned
6      for $k \leftarrow 1, 2, \ldots$ do                                           /* refinement loop */
7          $\hat{b} \leftarrow b - Ax_k$                                              /* compute violations */
8          $\hat{l} \leftarrow l - x_k$
9          $\hat{c} \leftarrow c - A^\mathsf{T}y_k$
10         $\delta_{P,k} \leftarrow \max\{\max_{j=1,\ldots,m} |\hat{b}_j|, \max_{i=1,\ldots,n} \hat{l}_i\}$          /* check termination */
11         $\delta_{D,k} \leftarrow \max\{0, \max\{-\hat{c}_i \mid i = 1, \ldots, n\}\}$
12         $\delta_{S,k} \leftarrow |\sum_{i=1,\ldots,n} -\hat{l}_i\hat{c}_i|$
13         if $\delta_{P,k} \leq \epsilon_P$ and $\delta_{D,k} \leq \epsilon_D$ and $\delta_{S,k} \leq \epsilon_S$ then
14             return $x^* \leftarrow x_k$, $y^* \leftarrow y_k$
15         $\Delta_{P,k+1} \leftarrow 1/\max\{\delta_{P,k}, (\alpha\Delta_{P,k})^{-1}\}$                /* solve transformed problem */
16         $\Delta_{D,k+1} \leftarrow 1/\max\{\delta_{D,k}, (\alpha\Delta_{D,k})^{-1}\}$
17         get $\bar{b}, \bar{l}, \bar{c} \approx \Delta_{P,k+1}\hat{b}, \Delta_{P,k+1}\hat{l}, \Delta_{D,k+1}\hat{c}$ in precision of the LP solver
18         solve $\min\{\bar{c}^\mathsf{T}x \mid \bar{A}x = \bar{b}, x \geq \bar{l}\}$ approximately
19         $\hat{x}, \hat{y} \leftarrow$ approximate primal–dual solution returned
20         $x_{k+1} \leftarrow x_k + \frac{1}{\Delta_{P,k+1}}\hat{x}$                                /* perform correction */
21         $y_{k+1} \leftarrow y_k + \frac{1}{\Delta_{D,k+1}}\hat{y}$
```

This process is repeated until the required accuracy is reached. All operations are performed in exact rational arithmetic unless otherwise noted.

To our knowledge, such an iterative refinement algorithm for linear programming has not been described in the literature. However, we want to mention the presentation of Saunders and Tenenblat (2006) on warm-starting interior point methods for convex quadratic programs via a "zoom strategy" that also shifts the problem and scales quadratic terms in the objective function. Furthermore, 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. In particular, we have heard that some interior point solvers may 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 (Ladányi 2011).

Remark 1. Classical iterative refinement for linear systems does not scale the residual errors on the right-hand side. Instead it exploits the fact that floating-point arithmetic is more accurate close to zero, so solving the linear system in floating-point naturally yields an error that is relative in the right-hand side. This does not hold for linear programming, since LP solvers work with fixed, absolute tolerances. If we did not scale the right-hand side, bounds, and reduced-cost values in the objective function of the transformed problem $\hat{P}$, a standard floating-point LP solver would consider the zero solution optimal within its tolerances. This would result in a zero corrector solution $\hat{x}, \hat{y}$ and no increase in precision. Furthermore, we need scaling for the same reason as in the iterative refinement scheme for linear systems of Wan (2006)—because we want to compute solutions that have a higher precision than can be represented in the precision of the floating-point solver.

3.2. Convergence

In the iterative refinement scheme of Algorithm 2, the approximate LP solver is treated as a black-box oracle. This permits an implementation where the LP solver is accessed through an interface and has the advantage that it allows substitution whenever an application benefits from a specific LP algorithm. The following basic assumption suffices to obtain a sequence of increasingly accurate solutions.

Assumption 1. *For every $A \in \mathbb{R}^{m\times n}$ there exist constants ϵ, $0 \le \epsilon < 1$, and $\sigma \ge 0$ such that for all $c, l \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$ for which the LP $\min\{c^\mathsf{T}x \mid Ax = b, x \ge l\}$ is primal and dual feasible, the LP solver returns an approximate primal-dual solution $\bar{x} \in \mathbb{Q}^n, \bar{y} \in \mathbb{Q}^m$ that satisfies*
1. $\|A\bar{x} - b\|_\infty \le \epsilon$
2. $\bar{x} \ge l - \epsilon\mathbb{1}$
3. $c - A^\mathsf{T}\bar{y} \ge -\epsilon\mathbb{1}$
4. $|\gamma(\bar{x}, \bar{y})| \le \sigma$

when it is given the LP $\min\{\bar{c}^\mathsf{T}x \mid \bar{A}x = \bar{b}, x \ge \bar{l}\}$, where $\bar{A} \in \mathbb{Q}^{m\times n}$, $\bar{c}, \bar{l} \in \mathbb{Q}^n$, and $\bar{b} \in \mathbb{R}^m$ are A, c, l, and b, respectively, rounded to the working precision of the LP solver.

This assumption suffices for the following simple convergence result.

Corollary 1. *Suppose we are given a primal and dual feasible LP as in (1) with constraint matrix A for which Assumption 1 holds with constants ϵ and σ. Let $x_k, y_k, \Delta_{P,k}$, and $\Delta_{D,k}$, $k = 1, 2, \ldots$, be the sequences of primal-dual solutions and scaling factors produced by Algorithm 2 with incremental scaling limit $\alpha > 1$, and let $\tilde{\epsilon} := \max\{\epsilon, 1/\alpha\}$. Then for all k,*
1. $\Delta_{P,k}, \Delta_{D,k} \ge 1/\tilde{\epsilon}^{k-1}$,
2. $\|Ax_k - b\|_\infty \le \tilde{\epsilon}^k$,
3. $x_k - l \ge -\tilde{\epsilon}^k\mathbb{1}$,
4. $c - A^\mathsf{T}y_k \ge -\tilde{\epsilon}^k\mathbb{1}$, and
5. $|\gamma(x_k, y_k)| \le \sigma\tilde{\epsilon}^{2(k-1)}$.

Hence, Algorithm 2 terminates after at most $\max\{\log(\epsilon_P), \log(\epsilon_D), \log(\epsilon_S\epsilon/\sigma)/2\}/\log(\tilde{\epsilon})$ approximate LP solves.

Proof. See Section A.2 of the online supplement. □

Several remarks on the reasonableness of Assumption 1 are in order. As Klotz (2014) points out for the CPLEX LP code, state-of-the-art LP solvers typically use an absolute definition for their tolerance requirements. First and foremost, because limited floating-point precision is by construction relative, an LP solver based only on floating-point computation will in general not be able to return solutions within absolute tolerances—certainly not the fast LP solvers we have in mind for practical applications. Otherwise, this would permit the following, much simpler approach. In Theorem 1, choose $x^*, y^* = 0$ and scaling factors $\Delta_P = \Delta_D = N$ arbitrarily large and solve ($\hat{P}$) for a solution $\bar{x}, \bar{y}$ as guaranteed by Assumption 1. Then $\bar{x}/N, \bar{y}/N$ would violate primal and dual feasibility by at most ϵ/N and complementary slackness by at most σ/N^2. One refinement step would suffice to reach arbitrary precision.

Because of this, the primal and dual scaling factors in Algorithm 2 are limited by the inverse of the maximum primal and dual violations, respectively. As a result, the largest entries in the right-hand side, lower bounds, and objective function vector of the transformed LP that correspond to violations of primal and dual feasibility become at most one in absolute value. For these variables and constraints, a relative tolerance requirement implies the absolute tolerance requirement of Assumption 1. However, note that when in x_k, y_k the lower bound is already satisfied for a variable, then its lower bound in the transformed LP may have a large absolute value; and if its reduced cost is already nonnegative, then the same holds for the transformed objective function coefficient.

Although there is no guarantee that a floating-point solver will produce solutions that are accurate to within an absolute tolerance, with our scaling strategy we expect that a modern LP solver will yield satisfactory results for most LPs 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 made to Algorithm 2 to incrementally boost the working precision of the LP solver when needed, as is done in Algorithm 1.

Note that if the floating-point solver encounters numerical difficulties, it may not only return a solution with large violations, but may even incorrectly conclude infeasibility or unboundedness. Section 4.1 describes a robust implementation of the iterative refinement scheme that, to the extent possible, tries to cope with such violations of Assumption 1.

3.3. Arithmetic Precision

Like many iterative refinement procedures for linear systems, Algorithm 2 is a mixed-precision procedure. This has the advantage that the most involved part—LP solving—is executed in fast floating-point arithmetic. Expensive rational arithmetic is only used for computing and scaling the new objective function, right-hand side, and bounds of the transformed LP, which amounts to two matrix-vector multiplications and a constant multiple of $n + m$ elementary operations per refinement round. Hence the number of elementary operations performed in rational arithmetic grows linearly with the number of nonzeros of the constraint matrix.

Nevertheless, rational arithmetic remains computationally expensive. Unlike with floating-point arithmetic, its cost is not constant per operation, but may increase if the encoding length of the corrected solution grows with increased accuracy. In this respect, one crucial modification to Algorithm 2 is to round the scaling factors $\Delta_{P,k}$ and $\Delta_{D,k}$ to powers of two. This helps keep the representation of the refined solution simple. It does not affect convergence, since for this only the order of magnitude of the scaling factors matters.

Finally we use exact rational arithmetic to allow the computation of arbitrarily precise solutions. If the goal is merely to reach a certain fixed level of accuracy, then it may be possible to replace rational by (sufficiently high) extended-precision arithmetic.

4. Handling Infeasibility and Unboundedness

For infeasible or unbounded LPs no approximately primal and dual feasible reference solution exists that can be refined. In this case, our goal is to construct a high-precision certificate of infeasibility or unboundedness.

4.1. Testing Feasibility

By Farkas lemma we know that an LP of form (1) is feasible if and only if the following auxiliary LP has optimal objective value of 0:

$$\max\{(b-Al)^{\mathsf{T}}y \mid A^{\mathsf{T}}y \le 0, (b-Al)^{\mathsf{T}}y \le 1\}. \tag{3}$$

The last inequality on the objective function ensures boundedness, and if the optimal objective value is nonzero, it is equal to 1. Feasible solutions to (3) with positive objective value serve as infeasibility certificates to the LP (1) and are often referred to as *Farkas proofs*. Because the zero solution is trivially feasible, this LP is primal and dual feasible and iterative refinement can be applied to compute an arbitrarily accurate Farkas proof.

To integrate this most seamlessly into one refinement scheme for LPs with unknown status, we will see that it is more suitable to consider the dual of (3), which reads

$$\min\{\tau \mid A\xi + (b-Al)\tau = (b-Al), \xi, \tau \ge 0\}.$$

Substituting $1-\tau$ for τ gives the more natural formulation

$$\max\{\tau \mid A\xi - (b-Al)\tau = 0, \xi \ge 0, \tau \le 1\}, \tag{4}$$

to which we will refer to as *feasibility LP*. The following lemma summarizes how solving this LP gives either a primal feasible solution or a Farkas proof of infeasibility for (1).

Lemma 1. *Suppose we are given an LP in equality form* (1)*; then the following hold*:

1. *The auxiliary LP* (4) *is primal and dual feasible.*
2. *The original LP* (1) *is feasible if and only if the auxiliary LP* (4) *has an optimal objective value of* 1.
3. *If the optimal objective value of* (4) *is less than* 1 *and* y^* *is an optimal dual solution vector for* (4)*, then* $-y^*$ *is a Farkas proof for the infeasibility of* (1).
4. *If* (ξ^*, τ^*), $\tau^* > 0$ *is an approximate optimal solution of* (4) *that violates primal feasibility by at most* ϵ_P*, then* $x^* = (1/\tau^*)\xi^* + l$ *is a feasible solution for* (1) *within tolerance* ϵ_P/τ^*.

Proof. See Section A.3 of the online supplement. □

As point 4 shows, when applying iterative refinement to the feasibility LP we have to adjust our termination criterion for primal feasibility. When a primal violation of $\delta_{P,k}$ is achieved in Algorithm 2, we may terminate if $\tau_k > 0$ and $\delta_{P,k}/\tau_k \le \epsilon_P$. If $\tau_k \approx 0$, the dual solution gives an approximate Farkas proof. For a discussion of how to test the feasibility of LPs in general form, see Section B.3 of the online supplement.

4.2. Computing a Rigorous Infeasibility Box from an Approximate Farkas Proof

Because a Farkas proof remains valid when multiplied by an arbitrary positive scalar, absolute tolerance requirements are meaningless. A Farkas proof of infeasibility for an LP in form (1) consists of a vector of dual multipliers $y \in \mathbb{Q}^m$ for the rows and a "reduced cost" vector $z \in \mathbb{Q}^n$ of multipliers for the bound constraints satisfying $z \ge 0$ such that the following holds:

$$A^{\mathsf{T}}y + z = 0 \tag{5}$$

$$b^{\mathsf{T}}y + l^{\mathsf{T}}z > 0. \tag{6}$$

An approximate Farkas proof may violate both of these conditions. Given an approximate Farkas proof y, z, any violations to Equation (5) can be corrected by adjusting z to be equal to $-A^{\mathsf{T}}y$. This, however, may set some components of z to negative values and may also create, or increase, a violation of (6). Even if iterative refinement applied to the feasibility LP produces Farkas proofs with smaller and smaller violations, this in general does not suffice to obtain a reliable certificate of infeasibility.

In the following, we introduce the concept of an *infeasibility box* that does not prove "approximate" infeasibility of the entire LP, but establishes exactly proven infeasibility within restricted bounds. As Neumaier and Shcherbina (2004) note, even if a Farkas proof is invalid in the classical sense of (5) and (6), the vector of dual multipliers y by aggregation gives the valid inequality

$$(y^{\mathsf{T}}A)x \ge b^{\mathsf{T}}y, \tag{7}$$

which we call a *Farkas cut*. If we can show that (7) is violated by all points x with $x \ge l$, then the LP is proven infeasible. In Neumaier and Shcherbina (2004) it was observed that interval arithmetic can be used to compute a lower bound on the left-hand side of (7), and if this is below the right-hand side value, it produces a certificate of infeasibility. However, this approach may fail, even if the approximate Farkas proof is very accurate.

We extend this notion by describing a method that will, given an approximate Farkas proof y, work backward to determine a domain in which no feasible solution x can exist. We compute the largest value R such that (7) is violated by all points x with $\|x\|_\infty < R$, i.e., that the feasible region (of the one-row relaxation given by the Farkas cut) intersected with the infeasibility box $\{x \mid -R\mathbb{1} < x < R\mathbb{1}\}$ is empty.

If the Farkas cut is written in form $d^\mathsf{T} x \geq d_0$, this largest R is computed by the mapping $\rho: \mathbb{Q}^n \times \mathbb{Q} \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ defined as

$$\rho(d, d_0) := \begin{cases} 0 & \text{if } d_0 \leq 0, \\ d_0/\|d\|_1 & \text{if } d \neq 0 \text{ and } d_0 > 0, \\ \infty & \text{if } d = 0 \text{ and } d_0 > 0. \end{cases} \tag{8}$$

If $\rho(d, d_0) = 0$, then the infeasibility box is empty; $\rho(d, d_0) = \infty$ implies that the full LP is successfully proven infeasible.

This kind of answer is both mathematically sound and helpful to users of an LP solver in practice. It allows them to conclude that feasible solution vectors must have large entries in absolute value, which might or might not be viable for the application at hand. The dimensions of this infeasibility box may be more comprehensible to an end user than, for example, a relative feasibility of a normalized Farkas proof.

Algorithm 3 describes the complete procedure for computing the infeasibility box. We assume that all arithmetic operations are executed in rational arithmetic. Throughout the algorithm, $d^\mathsf{T} x \geq d_0$ is a valid inequality for the LP and $R = \rho(d, d_0)$. First, the algorithm computes the aggregated constraint. Next, the loop starting on line 6 tries to incorporate the bounds on the variables to further increase the size of the infeasibility box. At this point, $d = A^\mathsf{T} y^*$, so $-d$ corresponds to the vector z in (5) and (6) of dual multipliers for the bound constraints. Suppose d_0 is positive and d contains more than one nonzero entry. For some $d_i < 0$, adding the inequality $-d_i x_i \geq -d_i l_i$ to $d^\mathsf{T} x \geq d_0$ increases the value of $\rho(d, d_0)$ if and only if

$$\begin{aligned} &\rho(d - d_i e_i, d_0 - d_i l_i) > \rho(d, d_0) \\ \Leftrightarrow\quad & \frac{d_0 - d_i l_i}{\sum_{q \neq i} |d_q|} > \frac{d_0}{\sum_q |d_q|} \\ \Leftrightarrow\quad & (d_0 - d_i l_i) \sum_q |d_q| > d_0 \sum_{q \neq i} |d_q| \\ \Leftrightarrow\quad & d_0 |d_i| - d_i l_i \|d\|_1 > 0 \\ \Leftrightarrow\quad & -l_i < \rho(d, d_0). \end{aligned} \tag{9}$$

This motivates the order in which the variables are considered when attempting to strengthen the Farkas cut in this manner; the variables with larger lower bounds are considered first.

At the end of the first loop, we may have $d_0 \leq 0$ and $R = 0$. As long as these hold, including lower bounds will increase d_0, although not necessarily R. Once $-l_i$ exceeds R, d_0 cannot be further increased by this procedure and the algorithm is terminated, returning R. Otherwise, we continue to include bound constraints in decreasing order as long as the criteria given by (9) are satisfied, returning R.

Algorithm 3 (Infeasibility box computation for an approximate Farkas proof)

```
   in: Inequality system $Ax = b$, $x \geq l$ with $A \in \mathbb{Q}^{m \times n}$,
       $b \in \mathbb{Q}^m$, $l, \in \mathbb{Q}^n$, approximate Farkas
       proof $y^* \in \mathbb{Q}^m$
   out: $R \geq 0$ such that $\|x\|_\infty \geq R$ for all feasible
        solutions $x$
 1 begin
 2     $d_0 \leftarrow b^\mathsf{T} y^*$
 3     $d \leftarrow A^\mathsf{T} y^*$
 4     $R \leftarrow \rho(d, d_0)$
 5     reindex variables such that $l_1 \geq l_2 \geq \cdots \geq l_n$
 6     for $i \leftarrow 1, 2, \ldots, n$ do
                           /* include bound constraints */
 7         if $-l_i \geq R$ then
 8             return $R$
 9         else if $d_i < 0$ then
10             $d_0 \leftarrow d_0 - d_i l_i$
11             $d_i \leftarrow 0$
12             $R \leftarrow \rho(d, d_0)$
13     return $R$
```

Remark 2 (Infeasibility Box Arithmetic). The infeasibility box algorithm as described uses rational arithmetic to obtain provable results. To save computation, R can be updated efficiently instead of recomputing $\rho(d, d_0)$ from scratch several times. Using the sparsity of the constraint matrix A and of vectors y^*, d is also critical. Still, rational arithmetic may in some cases be too expensive. Alternatively, Algorithm 3 could be implemented using interval arithmetic, which is faster and still yields proven results, if also potentially a smaller infeasibility box.

It remains to be explained how the computation of an infeasibility box interacts with iterative refinement of the feasibility LP. Intuitively, the size of the infeasibility box should grow as the approximate Farkas proof given as input becomes more accurate. This becomes most clear when we look at the simple case of a system of equations without bounds, $Ax = b$, with the feasibility LP $\max\{\tau \mid A\xi - b\tau = 0, \tau \leq 1\}$.

Let ξ_k, τ_k, y_k be a sequence of more and more accurate primal-dual solutions produced by Algorithm 2. From the proof of Theorem 1 it follows that not only the total violation of complementary slackness γ, but also the individual violation w.r.t. τ goes to 0; i.e.,

$$(1 - b^\mathsf{T} y_k)(1 - \tau_k) \to 0.$$

If τ_k does not converge to one, i.e., $\tau_k < C < 1$ (indicating infeasibility of the system), then $b^\mathsf{T} y_k \to 1$. At the same time, the dual violation $\|0 - A^\mathsf{T} y_k\|_\infty$ goes to 0. If we apply Algorithm 3 to y_k, then after the first loop we will have $d = A^\mathsf{T} y_k$ and $d_0 = b^\mathsf{T} y_k$. Hence, $R = d_0/\|d\|_\infty$ grows toward infinity with increasingly accurate iterates y_k.

This makes it natural to interleave the infeasibility box computation with the iterative refinement of the feasibility LP right after checking termination in line 13 of Algorithm 2. It will typically be called after a normal floating-point solve that claimed infeasibility, hence we assume an approximate Farkas proof as input that is tested at the beginning with Algorithm 3. If the computed radius of the infeasibility box is below the termination threshold, we continue to construct the feasibility LP by shifting bounds and sides. As a first reference point we choose the all zero solution for the primal and the given approximate Farkas proof for the dual solution.

The refinement loop works as in Algorithm 2 except when checking termination (which is skipped for the first artificial solution). If tolerance ϵ_P is not satisfied, this is either because (ξ_k, τ_k) is not yet sufficiently accurate for the feasibility LP or because the optimal value of the feasibility LP is less than 1. In the former case we continue with the next refinement step; in the latter case we run Algorithm 3 on y_k. The latter is checked by $\tau_k < 1 - \delta_{S,k+1}$ because careful calculation shows that for y_k the left-hand term in (6), which needs to be positive, equals $1 - \tau_k - \gamma(\xi_k, \tau_k, y_k)$. It is important that we do not use a fixed threshold to compare τ_k against one since the optimal value of the feasibility LP for infeasible LPs may be arbitrarily close to one. For discussion on how the infeasibility box is constructed for LPs in general form, see Section B.4 of the online supplement.

4.3. Testing Unboundedness

A certificate of primal unboundedness consists of a feasible primal solution vector and an unbounded direction of improving objective function value in the recession cone. The former can be computed as previously described. For the latter we apply the above feasibility test—assuming the LP is in form (1)—to the system

$$\begin{aligned} Av &= 0 \\ v &\geq 0 \\ c^{\mathsf{T}} v &= -1. \end{aligned} \tag{10}$$

Since the primal violation of an unbounded ray is not scale invariant, we should normalize the violation of primal feasibility by the objective function decrease; i.e., given an approximate unbounded direction v^*, we should use the violation of $v^*/|c^{\mathsf{T}} v^*|$ for checking the primal termination tolerance. This guarantees a maximum increase of the primal violation by ϵ_P as we decrease the objective function by one unit along the ray. Discussion on testing unboundedness for general form LPs can be found in Section B.5 of the online supplement.

4.4. An Integrated Refinement Algorithm

After discussing how iterative refinement is applied to infeasible and unbounded LPs separately, this section will show how these techniques are integrated into one algorithm to handle LPs of an a priori unknown status. A crucial property of such an algorithm is its ability to cope with incorrect answers of the underlying floating-point LP solver, since numerically challenging problems for which floating-point solvers return inconsistent results are one of the prime motivations for applying iterative refinement.

Figure 3 gives a flowchart of the integrated algorithm. We start by applying Algorithm 2 to the given LP. If the floating-point solver returns approximately optimal solutions at each call, the refinement steps will yield increasingly accurate solutions and we return a solution meeting the termination tolerances. If the floating-point solver concludes infeasibility or unboundedness for the initial or one of the transformed LPs, then the refinement loop is interrupted to test this claim. Here we implicitly rely on the fact that the transformed LP is infeasible or unbounded if and only if the original LP is infeasible or unbounded, respectively, which is a consequence of Theorem 1.

In the case of floating-point infeasibility, we apply iterative refinement to the feasibility LP (4) described in Section 4.1. Using the primal formulation here keeps the overhead of transforming the original LP low, whereas the dual formulation would result in increased overhead. We only need to add one column for the auxiliary variable τ to the constraint matrix, then modify the objective function and some of the bounds and sides. If this refinement terminates with an approximately feasible solution, we reject the floating-point solver's claim of infeasibility and start the iterative refinement algorithm again. Since the attempt to establish infeasibility has failed, we again attempt to compute an optimal solution, but do so with modified settings in the floating-point LP solver such as alternate tolerance levels and hope for success. Otherwise, we terminate and conclude infeasibility.

If the floating-point solver claims unboundedness, we first apply iterative refinement to (10) to compute a primal ray. If this fails, we restart the refinement of the original LP with modified floating-point parameters. If it succeeds, we continue testing feasibility. If we obtain a primal feasible solution, we conclude unboundedness. Otherwise, we return (primal and dual) infeasibility.

Remark 3. Some LPs are formulated such that a small perturbation may change the feasibility status of the problem. As a result, there may simultaneously exist solutions that are feasible within a small feasiblity tolerance as well as highly accurate Farkas proofs. LPs near or on this boundary of feasibility and infeasibility are called ill conditioned or ill posed (see Renegar 1994).

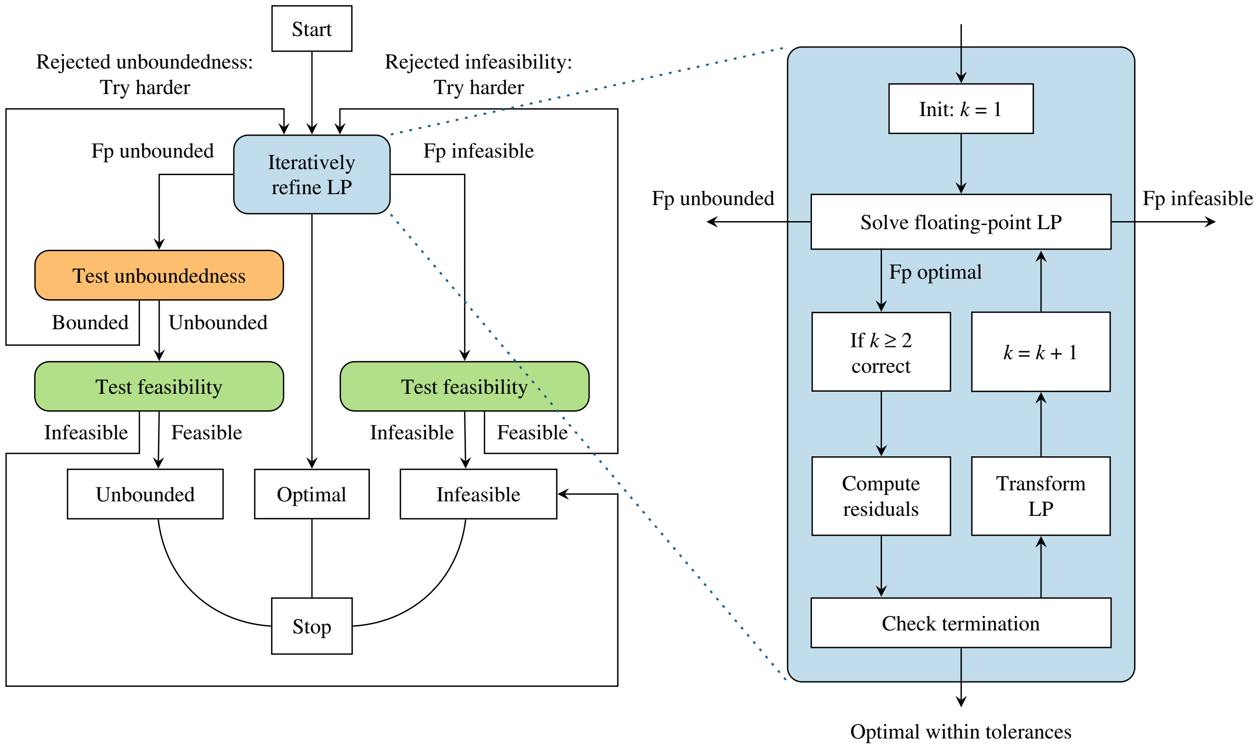

Figure 3 (Color online) Iterative Refinement for LPs of a priori Unknown Status

As our algorithm first attempts to find and refine a feasible solution, it is in a sense biased toward finding approximate feasible solutions if they exist, even if approximate Farkas proofs also exist. In such cases one may at least conclude that a small perturbation of the LP would be feasible.

5. Computational Study

As pointed out, Assumption 1 regarding on the accuracy of the floating-point solutions will not hold in general. We hope that in practice it is satisfied for LPs constructed during iterative refinement. In the following, we describe computational results for a simplex-based implementation. One of the motivations for basing our experiments on a simplex solver instead of an interior point solver is the fact that the LPs solved are highly similar in that they all share the same constraint matrix and their solution spaces are affine transformations of each other. Hence the solution refined by the last LP solved gives a starting point for the next LP and its basis information carries over, as shown by Theorem 1. Although modern implementations of interior point methods are reported to be faster on average for solving LPs from scratch, the unmatched hot-starting[3] capabilities of the simplex method promise greater gains in performance when a larger number of refinement rounds are applied.

5.1. Implementation

Starting with Version 1.7 released in July 2012, we extended the SoPlex[4] LP solver (Wunderling 1996) with functionality to read, store, check, and process LPs and LP solutions in rational precision. Our implementation is based on the GNU Multiple Precision Arithmetic Library[5] and can be linked to EGlib[6] for faster memory allocation.

5.1.1. Using Basis Information. The basis information is first used to compute the maximum dual violation $\delta_{D,k}$ in line 11 of Algorithm 2. The reduced cost of a variable is considered infeasible if it is positive but the basis status of the column is *not* nonbasic (i.e., not fixed) at the lower bound. The violation of complementary slackness is not tracked and line 12 is skipped since basic solutions, by construction, satisfy complementary slackness.

Second, we hot start the LP solver in line 18 from the previous basis, reusing the factorization of the basis matrix. Note that unless the primal and dual

[3] *Hot starting* is an advanced form of warm starting where, in addition to basis information being reused, the basis matrix factorization is also reused and updated, reducing the time spent on linear algebra subroutines.

[4] Zuse Institute Berlin. SoPlex—the Sequential Object-oriented simPlex. http://soplex.zib.de/.

[5] http://www.gmplib.org/.

[6] Daniel G. Espinoza and Marcos Goycoolea. EGlib. Efficient General Library. http://www.dii.uchile.cl/~daespino/EGlib_doc/main.html.

scaling factors are limited by the incremental scaling limit α, the corresponding basic solution is guaranteed to be primal and dual infeasible for the transformed LP. SoPlex allows for arbitrary regular starting bases using a shifting technique (see Wunderling 1996, Section 1.4).

Third, we have to consider basis information in the correction step. Suppose in the floating-point solution $\hat{x}$ some variable is nonbasic at its lower bound, say $\hat{x}_i = \bar{l}_i$. Then the corrected solution value $(x_{k+1})_i$ may differ from l_i by $(\bar{l}_i/\Delta_{D,k+1}) - \hat{l}_i$. This is because of the necessary rounding step in line 17. We correct this by setting $\hat{x}_i$ directly to l_i in this case, which has the added advantage of avoiding one multiplication and addition in rational arithmetic. This adjustment of the corrector solution is on a tiny order of magnitude and should not compromise Assumption 1—if it does, then the floating-point solutions are almost certainly already unreliable.

5.1.2. Robust Floating-Point Solves. For numerically difficult instances, SoPlex may encounter numerical troubles and fail to solve one of the floating-point LPs—running into a singular basis, aborting as a result of cycling, returning infeasible or unbounded for the auxiliary LPs when testing feasibility or unboundedness, or returning infeasible or unbounded, although these results have been rejected earlier in the integrated solving loop displayed in Figure 3. If this happens, we try again with a series of different parameter settings until successful. If this still does not help, we terminate without reaching the desired tolerances.

5.1.3. Updating Residual Vectors. In line 7 of Algorithm 2 for $k \geq 2$, we can use $x_k = x_{k-1} + (1/\Delta_{P,k})\hat{x}$ to compute $b - Ax_k$ by subtracting $(1/\Delta_{P,k})A\hat{x}$ from the previous $\hat{b}$. If the corrector solution $\hat{x}$ is sparser than the corrected solution x_k, then this update is likely to be faster than recomputing $\hat{b}$ from scratch. The same holds for computing the new objective function in line 9. We already store the differences $(1/\Delta_{P,k+1})\hat{x}$ and $(1/\Delta_{D,k+1})\hat{y}$ as sparse vectors when correcting the primal and dual solution in lines 20 and 21.

This is a heuristic minimization of the number of arithmetic operations performed in rational arithmetic since we do not take into account a potentially unequal distribution of nonzeros over the constraint matrix. However, it promises a good approximation. Especially when the basis between two subsequent floating-point solves does not change much or at all, which is typical after a few refinement rounds, the primal update vectors will contain only entries for basic variables; as a result, they will typically be much sparser than the corrected solution.

5.2. Setup

The goal of our experiments was threefold. First, we wanted to analyze the behavior of the plain iterative refinement procedure for computing high-precision solutions. Does it always converge and how fast does it converge, both in terms of the number of refinements and the time spent? How much time is spent in the refinement phase compared to the initial floating-point solve? How expensive are the rational arithmetic operations performed? Are there differences between numerically easy and difficult instances?

To this end, we performed the plain floating-point solve of standard SoPlex without iterative refinement with a primal and dual tolerance of 10^{-9} and compared it to iterative refinement runs with both primal and dual tolerances of 10^{-50} and 10^{-250}. We will denote these settings by SoPlex$_9$, SoPlex$_{50}$, and SoPlex$_{250}$, respectively. In all cases, we parsed LP files exactly and computed violations in rational arithmetic.

Second, we wanted to know whether the iterative refinement procedure always converges to an optimal basis and how many refinements are needed? For this, we used the basis verification tool perPlex of Koch (2004).

Third, we wanted to investigate whether iterative refinement helps improve the performance of exact LP solving. A natural idea to test this is to warm start the QSopt_ex solver—which today may be considered as the state of the art in general-purpose exact LP solving—from the advanced starting basis obtained after several rounds of iterative refinement, at the end of SoPlex$_{50}$, say, and measuring the total running time of SoPlex$_{50}$ and QSopt_ex. Comparing this to the plain QSopt_ex performance would be biased, however, since the underlying floating-point simplex implementations in SoPlex and QSopt_ex are significantly different. (QSopt_ex, e.g., does not implement presolving techniques.) As a meaningful point of reference we use the performance of QSopt_ex when warm started from the basis returned by SoPlex$_9$. We measured whether iterative refinement decreases the running time and maximum precision used by QSopt_ex.

Last, we need to comment on the parameter settings used for the floating-point LP solves. By default, SoPlex uses an absolute feasibility tolerance of 10^{-6}, But in our experiments, we have tightened it to 10^{-9}. Generally, using a stricter tolerance will return higher-precision corrector solutions, but at the same time too strict a tolerance can lead to a numerical breakdown, which is why we did not use a value of 10^{-12}. Presolving and scaling were applied when solving LPs from scratch, but not when hot starting from an advanced basis.

5.2.1. Hardware and Software. The experiments were conducted on a cluster of 64-bit Intel Xeon X5672 CPUs at 3.2 GHz with 12 MB cache and 48 GB main memory, simultaneously running only one job per node. We used a time limit of two hours per instance for each SoPlex, perPlex, and QSopt_ex run. We used the SoPlex developer version 2.0.0.2, implementing the iterative refinement algorithm with features and parameters as described earlier. SoPlex was compiled with GCC 4.8.2 and linked to the external libraries GMP 5.1.3, EGlib 2.6.20, and zlib 1.2.8 for reading compressed instance files.

5.2.2. Instances. We compiled a large test set of 1,202 primal and dual feasible LPs from several sources: the Netlib LP test set, including the *kennington* folder, Hans Mittelmann's benchmark instances, Csaba Mészáros's LP collection, the LP relaxations of the COR@L mixed-integer programming test set, and the LP relaxations of the mixed-integer programs from MIPLIB, MIPLIB 2, MIPLIB 3, MIPLIB 2003, and MIPLIB 2010. A detailed description of this collection with problem statistics is given in Table 4 in Section C of the online supplement.

5.3. Computational Results

Let us first look at two individual instances, `momentum3` from MIPLIB 2003 and `world` from Mészáros's test set. These are nontrivial instances with 949,495 nonzeros and 164,470 nonzeros in their constraint matrix, respectively. Table 1 shows how the primal and dual violations progress with time and number of simplex iterations elapsed. Because we are mainly interested in the order of magnitude of the violations, we only report the precision as the rounded negative base 10 logarithm.

Instance `momentum3` shows steady convergence. Each iteration added between seven and 16 orders of magnitude to the precision of the solution, such that after 18 refinement rounds primal and dual violations below 10^{-250} are reached. The initial floating-point solve with 46,184 iterations consumes the largest part of the running time. Only during the first refinement LP is one more simplex iteration performed. After that, the basis remains unchanged. The refinement phase only incurs an overhead of 4.2% in running time and is dominated by rational arithmetic. The hot start proves very efficient, and the simplex time for solving the refinement LPs is negligible. The time consumed by rational arithmetic increases only slightly as the precision of the solution improves, from 0.3 seconds for the first to 0.5 seconds for the last refinement. This is largely a result of the positive effect of rounding the scaling factors to powers of two; in an earlier implementation without this feature, the last refinements consumed almost two seconds. When refining further, this slowdown became increasingly pronounced.

The observed distribution of running time is typical and similar for the instance `world`. Numerically, however, it seems to be more challenging to SoPlex and the convergence of iterative refinement is slower. Simplex pivots, although comparatively few, are performed up to refinement round 12 until the final basis is reached. In the rounds where pivots are performed, the dual violation does not decrease, meaning that the floating-point corrector solutions returned by SoPlex must exhibit high absolute violation in dual multipliers or reduced costs. This violation of Assumption 1 is explained as follows. In the transformed LP, nonbasic variables with (dual) feasible reduced cost may have large objective coefficients, because they do not limit

Table 1 **Progress of Iterative Refinement for Instances `momentum3` and `world`**

	`momentum3`					`world`				
R	iter	t	t_{rat}	δ_P	δ_D	iter	t	t_{rat}	δ_P	δ_D
0	46,184	190.4	0.3	8	10	70,204	131.5	0.1	8	11
1	46,185	191.0	0.6	23	17	70,256	131.7	0.2	17	12
2	46,185	191.3	1.0	37	33	70,282	131.9	0.4	22	13
3	46,185	191.7	1.3	51	48	70,287	132.1	0.5	23	13
4	46,185	192.0	1.7	65	65	70,289	132.3	0.7	23	14
5	46,185	192.4	2.1	80	78	70,292	132.4	0.9	24	13
6	46,185	192.8	2.4	93	93	70,292	132.6	1.0	24	29
7	46,185	193.2	2.8	108	107	70,315	132.8	1.2	34	13
8	46,185	193.6	3.2	121	120	70,315	133.0	1.4	34	29
9	46,185	194.0	3.6	136	134	70,319	133.2	1.6	40	13
10	46,185	194.4	4.0	150	149	70,319	133.4	1.7	40	29
11	46,185	195.1	4.7	165	164	70,319	133.6	1.9	40	45
12	46,185	195.6	5.1	178	177	70,320	133.8	2.1	55	15
13	46,185	196.0	5.6	193	191	70,320	134.0	2.3	70	30
14	46,185	196.4	6.0	207	205	70,320	134.2	2.5	85	46
15	46,185	196.9	6.4	222	220	70,320	134.4	2.7	100	61
16	46,185	197.4	6.9	235	234	70,320	134.6	2.9	115	77
17	46,185	197.9	7.4	250	248	70,320	134.8	3.1	130	93
18	46,185	198.4	7.9	264	262	70,320	135.0	3.3	146	107

Note. R—number of refinements, iter—number of simplex iterations elapsed, t—time elapsed (in seconds), t_{rat}—time spent on rational arithmetic (cumulative, in seconds), δ_P—maximum primal violation (rounded negative $\log_{10}$), δ_D—maximum dual violation (rounded negative $\log_{10}$).

the dual scaling factor. If these are pivoted into the basis, their reduced cost should be zero. Solving the linear system for the dual solution vector, however, only yields a precision that is relative with respect to the (large) objective coefficients.

As we can see in Table 1, after the first refinement round without pivots, round six, the dual violation starts to decrease below 10^{-29}. In the next round, however, new pivots occur, and the maximum dual violation even falls back to 10^{-13}. From round 13 on both primal and dual precision improves continuously, in each round reducing the violations by about 15 orders of magnitude. Because of the setbacks during the refinement rounds with simplex pivots, the dual precision lags behind the primal precision. This could be even more pronounced, but because we limit the primal scaling factor by the dual scaling factor in our implementation, the primal precision stalls during some rounds. After 29 refinement rounds (not shown in the table), a maximum violation below 10^{-250}, comparable to the precision on `momentum3` after 18 rounds, is reached.

5.3.1. General Results. We move on to discuss the results over the entire test set. Detailed results for each instance can be found in Table 5 in Section C of the online supplement. Out of the 1,202 instances in our test set, SoPlex$_{50}$ and SoPlex$_{250}$ converged successfully to the specified tolerance on 1,195 instances. On three instances,[7] they timed out because the floating-point solver could not solve the first refinement LP within the time limit. For three instances, the initial floating-point solve (equivalent to SoPlex$_{9}$) incorrectly claimed unboundedness, and for one instance it incorrectly claimed infeasibility.[8] In all cases, SoPlex$_{50}$ and SoPlex$_{250}$ rejected these claims successfully using the feasibility and unboundedness tests described in Section 4.1, but after starting to refine the original LP again, floating-point SoPlex failed to return an approximately optimal solution even when run with different settings. Furthermore, for five of the 1,195 instances[9] the floating-point solver claimed one of the intermediate refined LPs infeasible. SoPlex$_{50}$ and SoPlex$_{250}$ rejected these claims and continued to converge to their target tolerance.

5.3.2. Results for the Floating-Point Solves. For four instances SoPlex$_{9}$ claimed a wrong status,[8] and for 99 instances it returned a numerical solution that exhibited violations *above* 10^{-9}, though most of those only slightly. However, for the instance `de063155` from Mészáros's "problematic" test set, which features constraint coefficients with absolute values ranging from approximately 10^{-7} to 10^{12}, SoPlex$_{9}$ even returned a completely meaningless solution, violating primal and dual feasibility by almost 10^{3} and 10^{8}, respectively. This instance is solved correctly by SoPlex$_{50}$ and SoPlex$_{250}$ in seven and 20 refinements, respectively.

For 1,024 instances perPlex verified that the basis returned by SoPlex$_{9}$ was indeed optimal. For 95 instances, the SoPlex$_{9}$ basis was detected as primal infeasible, for 30 instances as dual infeasible, and for six instances as both primal and dual infeasible. On the remaining instances, perPlex hit a time or memory limit, or—as for the instance `neos-619167`—it could not handle free nonbasic variables correctly. Hence, our test set seems to contain a nonnegligible number of numerically challenging instances. Although for most instances, we can confirm the conclusion of Dhiflaoui et al. (2003) and Koch (2004) that floating-point LP solvers often succeed in returning optimal bases (on the Netlib test set), we have to relativize this finding for more than 14% of the instances.

5.3.3. Results for Iterative Refinement. As previously mentioned, on all 1,195 instances, both SoPlex$_{50}$ and SoPlex$_{250}$ converged to the specified tolerance. On all instances that perPlex could handle, it verified that the final basis returned was primal and dual feasible. For 61 instances, they even terminated with an exactly optimal numeric solution with a zero primal and dual violation.

For the vast majority of instances, the basis after the initial floating-point solve was already the final basis later returned. On 93 and 30 instances, one and two refinement rounds, respectively, were needed to reach the final (optimal) basis. Only for instances `de063155`, `mod2`, and `world`, three, 11, and 12 rounds, respectively, were performed until the final basis. This happened when using SoPlex$_{50}$, so the additional rounds performed by SoPlex$_{250}$ essentially only refine the primal and dual solutions to the linear systems defined by the basis matrix. Furthermore, for the five instances[9] for which feasibility was tested, only few refinements with actual pivots were performed.

5.3.4. Average Performance Comparison. To compare performance of the three runs on subsets of instances with varying numerical difficulty, we categorized the instances according to the number of refinement rounds performed until the final basis was reached, denoted by R_0. We excluded the five instances[9] where the infeasibility test was triggered, because their run actually amounts to the refinement of two LPs. We excluded simple instances that were solved in under two seconds by each algorithm.

For the resulting 356 instances, Table 2 reports the average number of refinement rounds, simplex iterations, and average running time over these subsets.

[7] `neos-954925`, `neos-956971`, and `neos-957143`.

[8] `de063157`, `l30`, and `stat96v1` were claimed to be unbounded; `neos-1603965` was claimed to be infeasible by SoPlex$_{9}$.

[9] `fome11,12,13`, `rail01`, and `shs1023`.

Table 2 Computational Comparison of Iterative Refinement and Pure Floating-Point Performance

		SoPlex$_9$		SoPlex$_{50}$				SoPlex$_{250}$			
R_0	N	iter	t	R	iter	t	Δt	R	iter	t	Δt
0	326	19,291.8	20.0	3.0	19,291.8	20.6	1.03	16.4	19,291.8	22.7	1.14
1	26	16,936.8	20.6	3.9	17,918.2	22.3	1.08	17.8	17,918.2	25.1	1.22
2	2	79,895.4	90.3	4.5	79,964.6	92.8	1.03	17.5	79,964.6	99.0	1.10
11	1	58,340.0	100.4	14.0	58,534.0	103.8	1.03	26.0	58,534.0	104.2	1.04
12	1	70,204.0	131.2	15.0	70,320.0	132.1	1.01	29.0	70,320.0	137.6	1.05
≥ 1	30	20,531.5	25.7	4.6	21,562.4	27.6	1.07	18.5	21,562.4	30.7	1.19

Note. R_0—number of refinements to final basis, N—number of instances in this R_0-class, iter—number of simplex iterations (shifted geometric mean), t—total running time (shifted geom. mean in seconds), R—number of refinements (arithmetic mean), Δt—relative running time with respect to SoPlex$_9$.

Because simplex iterations and running times vary drastically across instances, we computed their averages not as arithmetic means—which would introduce a bias toward large values—but as shifted geometric means. We use shifts of two seconds and 100 simplex iterations. For the number of refinements we report arithmetic averages.

For SoPlex$_{50}$, most notably, the average running time increases by only 3% for the easy subset $R_0 = 0$ and by only 7% for the set $R_0 \geq 1$ of instances with at least one refinement round with simplex pivots. This is reflected in the small number of additional simplex iterations that are performed during the refinement rounds. Furthermore, the average number of refinement rounds is often smaller than expected. To achieve a maximum violation of 10^{-250} by floating-point solves with tolerance 10^{-9}, one would have to estimate approximately $250/9 - 1 \approx 26.8$ refinement rounds. By contrast, even the class $R_0 \geq 1$ shows only 18.5, indicating that most floating-point solves—in particular the final ones—return higher-precision solutions. This is slightly less pronounced for SoPlex$_{50}$.

The overhead in the running time of SoPlex$_{250}$ beyond SoPlex$_{50}$ stems only from refining the primal and dual solution to the linear systems defined by the basis matrix. We note that the increase in time of SoPlex$_{250}$ to SoPlex$_9$ is still very small on average—14% for $R_0 = 0$ and 19% for $R_0 \geq 1$. Incorporating some of the more sophisticated techniques recently developed by Wan (2006), Pan (2011), or Saunders et al. (2011) may close this gap even further.

5.3.5. Accelerating Exact LP. Finally, we compared the performance of QSopt_ex when warm started from SoPlex$_9$ and SoPlex$_{50}$, respectively. Of the 1,195 instances for which both SoPlex$_9$ and SoPlex$_{50}$ returned basic solutions, 1,166 instances could be solved by both versions. Nine instances could be solved only when warm starting from the advanced basis returned by SoPlex$_{50}$ (within running times between 108 and 2,026 seconds), whereas starting from SoPlex$_9$'s basis QSopt_ex hit the time limit of two hours.[10] Twenty instances could not be solved by either version. Detailed results can be found in Table 6 in Section C of the online supplement.

Table 3 gives an aggregated comparison over the 307 instances that were solved by both versions but that were nontrivial in the sense that at least one version took more than two seconds. It compares the total number of iterations taken (by SoPlex and QSopt_ex) and the total running time, and additionally states the running times of SoPlex and QSopt_ex individually. We again report shifted geometric means, using a shift of two seconds and 100 simplex iterations. (Note that it is hence correct that the values in columns $t_{9/50}$ and t_{ex} do not exactly add up to the t-values.) To distinguish numerically easy from difficult instances, we again categorized them by the number of refinement rounds (R_0) needed by SoPlex$_{50}$ to reach the final basis.

On the instances for which iterative refinement did not change the basis, the QSopt_ex performance is necessarily identical. It never increased the simplex precision beyond 64-precision except for the instance `maros-r7`, for which it started pivoting despite the optimality of the starting basis and performed to precision boosts to 192-precision. The total running time on these instances increases by only 1% on average, corresponding to the small overhead of iterative refinement. In contrast, for the 23 instances for which iterative refinement affected the final basis, the performance gain is drastic: the total running time is reduced to only 18%, i.e., by a factor 5.5 on average. (Note that this does not even include the nine instances[10] on which SoPlex$_9$+QSopt_ex timed out.) This improvement cannot be explained by the reduction in simplex iterations alone. Most importantly, with warm starting from the basis returned by SoPlex$_{50}$, no precision boosts were performed, so the expensive pivots performed by QSopt_ex in extended-precision were reduced to zero.

[10] `fome13` (in 328.6 seconds), `mod2` (in 108.1 seconds), `momentum3` (in 2025.5 seconds), `ofi` (in 432.2 seconds), `sgpf5y6` (in 228.7 seconds), `shs1023` (in 625.4 seconds), `watson_1` (in 379.8 seconds), `watson_2` (in 1538.1 seconds), and `world` (in 136.6 seconds).

Table 3 **Computational Comparison of QSopt_ex's Performance when Warm Started from Bases Returned by SoPlex$_9$ and SoPlex$_{50}$**

		SoPlex$_9$ + QSopt_ex					SoPlex$_{50}$ + QSopt_ex					
R_0	N	iter	B	t_9	t_{ex}	t	iter	B	t_{50}	t_{ex}	t	Δt
0	284	22,306.5	2	20.7	3.1	27.2	22,306.5	2	21.2	3.1	27.5	1.01
1	22	23,812.9	13	18.1	110.2	168.7	16,423.4	0	19.4	4.6	31.0	0.18
2	1	51,469.0	1	18.8	240.8	259.6	45,950.0	0	19.7	2.6	22.2	0.09
≥ 1	23	24,625.4	14	18.1	114.0	171.9	17,176.3	0	19.4	4.5	30.6	0.18

Note. R_0—number of refinements to final basis, N—number of instances in this R_0-class, iter—number of simplex iterations by SoPlex + QSopt_ex (shifted geom. mean), B—number of precision boosts in QSopt_ex (total over all LPs in this R_0-class), $t_{9/50}$—running time of SoPlex$_{9/50}$ (shifted geom. mean in seconds), t_{ex}—running time of QSopt_ex (shifted geom. mean in seconds), t—total running time (shifted geom. mean in seconds), Δt—relative total running time w.r.t. SoPlex$_9$ + QSopt_ex.

6. Conclusion

We have presented a new algorithm to solve linear programs to high levels of precision. It extends the idea of iterative refinement for linear systems of equations by Wilkinson (1963) to the domain of optimization problems by simultaneously correcting primal and dual residual errors. Algebraically, it builds up an increasingly accurate solution by solving a sequence of LPs, which differ only in the bounds of the variables, the sides of the constraints, and the objective function coefficients, to fixed precision. Geometrically, it can be viewed as zooming further and further into the area of interest around the refined solution. Although it is designed to work with an arbitrary LP oracle, it combines especially well with the hot-starting capabilities of the simplex method. For infeasible and unbounded instances it can be used to compute high-precision certificates via an auxiliary reformulation of the LP. In this context, we have developed an algorithm to convert an approximate Farkas proof into a rigorous infeasibility box centered at the origin that helps users understand the domains in which feasible solutions can or cannot exist.

For a simplex-based implementation we demonstrated it to be efficient in practice: on a large test set of publicly available benchmark instances, computing solutions up to a precision of 10^{-50} incurred an average slowdown of only 3% on numerically easy and 7% on numerically difficult LPs. In addition, we saw that the basis corresponding to the refined solution was always optimal. We exploited this to warm start the exact LP solver QSopt_ex. As a result, we observed a more than five times speedup for difficult instances and could solve nine more instances than QSopt_ex alone.

As with classical iterative refinement, the algorithm shares the limitation that it breaks down when the LP is too ill conditioned for the underlying floating-point routine. This could be overcome by increasing the working precision of the underlying floating-point LP solver whenever necessary in a similar fashion as in the exact LP solver QSopt_ex discussed in Section 2.1.

As a final remark, we note that some applications may require extended-precision solutions but not need exact solutions. In such cases iterative refinement can be used to meet this demand without requiring the amount of time taken by an exact LP solver, giving it a competitive advantage.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2016.0692.

Acknowledgments

Ambros M. Gleixner was supported by the Research Campus Modal *Mathematical Optimization and Data Analysis Laboratories* funded by the German Ministry of Education and Research [BMBF Grant no. 05M14ZAM]. Kati Wolter was supported by the DFG Priority Program 1307 Algorithm Engineering.

References

Althaus E, Dumitriu D (2009) Fast and accurate bounds on linear programs. Vahrenhold J, ed., *Proc. 8th Internat. Sympos. Experiment. Algorithms*, Lecture Notes in Computer Science, Vol. 5526 (Springer, Berlin), 40–50.

Applegate DL, Cook W, Dash S, Espinoza DG (2007a) Exact solutions to linear programming problems. *Oper. Res. Lett.* 35(6):693–699.

Applegate DL, Cook W, Dash S, Espinoza DG (2007b) QSopt_ex. Accessed April 19, 2016, http://www.dii.uchile.cl/~daespino/ESolver_doc/.

Avis D, Fukuda K (1992) A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geometry* 8(1):295–313.

Azulay D-O, Pique J-F (1998) Optimized *Q*-pivot for exact linear solvers. Maher M, Puget J-F, eds. *Principles and Practice of Constraint Programming CP98*, Lecture Notes in Computer Science, Vol. 1520 (Springer, Berlin), 55–71.

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

Bulutoglu DA, Kaziska DM (2010) Improved WLP and GWP lower bounds based on exact integer programming. *J. Statist. Planning Inference* 140(5):1154–1161.

Burton BA, Ozlen M (2012) Computing the crosscap number of a knot using integer programming and normal surfaces. *ACM Trans. Math. Software* 39(1):4:1–4:18.

Chindelevitch L, Trigg J, Regev A, Berger B (2014) An exact arithmetic toolbox for a consistent and reproducible structural analysis of metabolic network models. *Nature Comm.* 5:Article no. 4893.

Chvátal V (1983) *Linear Programming* (W. H. Freeman and Company, New York).

Cohn H, Jiao Y, Kumar A, Torquato S (2011) Rigidity of spherical codes. *Geometry Topology* 15(4):2235–2273.

Cook W, Steffy DE (2011) Solving very sparse rational systems of equations. *ACM Trans. Math. Software* 37(4):39:1–39:21.

Cook W, Koch T, Steffy DE, Wolter K (2011) An exact rational mixed-integer programming solver. Günlük O, Woeginger G, eds. *Integer Programming and Combinatoral Optimization*, Lecture Notes in Computer Science, Vol. 6655 (Springer, Berlin), 104–116.

Dantzig GB (1963) *Linear Programming and Extensions* (Princeton University Press, Princeton, NJ).

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

Dhiflaoui M, Funke S, Kwappik C, Mehlhorn K, Seel M, Schömer E, Schulte R, Weber D (2003) Certifying and repairing solutions to large LPs: How good are LP-solvers? *Proc. 14th Ann. Sympos. Discrete Algorithms, SODA '03* (SIAM, Philadelphia), 255–256.

Edmonds J (1994) Exact pivoting. Presentation, ECCO VII, May 26–28, Milan, Italy.

Edmonds J, Maurras J-F (1997) Note sur les Q-matrices d'Edmonds. *RAIRO. Recherche Opérationnelle* 31(2):203–209.

Espinoza DG (2006) On linear programming, integer programming and cutting planes. Unpublished doctoral dissertation, Georgia Institute of Technology, Athens, GA.

Fukuda K, Prodon A (1996) Double description method revisited. Deza M, Euler R, Manoussakis I, eds. *Combinatorics and Computer Science*, Lecture Notes in Computer Science, Vol. 1120 (Springer, Berlin), 91–111.

Gärtner B (1999) Exact arithmetic at low cost—A case study in linear programming. *Comput. Geometry* 13(2):121–139.

Gleixner AM, Steffy DE, Wolter K (2012) Improving the accuracy of linear programming solvers with iterative refinement. *Proc. 37th Internat. Sympos. Symbolic Algebraic Comput.* (ACM, New York), 187–194.

Golub G, van Loan C (1983) *Matrix Computations* (Johns Hopkins University Press, Baltimore).

Grötschel M, Lovasz L, Schrijver A (1988) *Geometric Algorithms and Combinatorial Optimization* (Springer-Verlag, Berlin).

Hales TC (2005) A proof of the Kepler conjecture. *Ann. Math.* 162(3):1065–1185.

Held S, Cook W, Sewell E (2012) Maximum-weight stable sets and safe lower bounds for graph coloring. *Math. Programming Comput.* 4(4):363–381.

Hicks I, McMurray N (2007) The branchwidth of graphs and their cycle matroids. *J. Combinatorial Theory Ser. B* 97(5):681–692.

Jansson C (2004) Rigorous lower and upper bounds in linear programming. *SIAM J. Optim.* 14(3):914–935.

Khachiyan LG (1979) A polynomial algorithm in linear programming. *Soviet Mathematics Doklady* 20(1):191–194. [Translation.].

Klotz E (2014) Identification, assessment and correction of ill-conditioning and numerical instability in linear and integer programs. Newman A, Leung J, eds. *TutORials in Operations Research: Bridging Data and Decisions* (INFORMS, Catonsville, MD), 54–108.

Koch T (2004) The final NETLIB-LP results. *Oper. Res. Lett.* 32(2): 138–142.

Ladányi L (2011) Personal communication, November 26. IBM T.J. Watson Research Center, Yorktown Heights, NY.

Lerman JA, Hyduke DR, Latif H, Portnoy VA, Lewis NE, Orth JD, Schrimpe-Rutledge AC, et al. (2012) In silico method for modelling metabolism and gene product expression at genome scale. *Nature Comm.* 3:Article no. 929.

Maes C (2013) Personal communication, September 6. Gurobi Optimization, Inc., Houston, TX.

Moore RE, Kearfott RB, Cloud MJ (2009) *Introduction to Interval Analysis* (Society for Industrial and Applied Mathematics, Philadelphia).

Neumaier A, Shcherbina O (2004) Safe bounds in linear and mixed-integer linear programming. *Math. Programming* 99(2):283–296.

Pan VY (2011) Nearly optimal solution of rational linear systems of equations with symbolic lifting and numerical initialization. *Comput. Math. Appl.* 62(4):1685–1706.

Renegar J (1994) Some perturbation theory for linear programming. *Math. Programming* 65(1):73–91.

Saunders BD, Wood DH, Youse BS (2011) Numeric-symbolic exact rational linear system solver. *Proc. 36th Internat. Sympos. Symbolic Algebraic Comput., ISSAC '11* (ACM, New York), 305–312.

Saunders MA, Tenenblat L (2006) The zoom strategy for accelerating and warm-starting interior methods. Presentation, INFORMS Annual Meeting, Pittsburgh.

Schrijver A (1986) *Theory of Linear and Integer Programming* (Wiley, Chichester, UK).

Steffy DE, Wolter K (2013) Valid linear programming bounds for exact mixed-integer programming. *INFORMS J. Comput.* 25(2): 271–284.

Ursic S, Patarra C (1983) Exact solution of systems of linear equations with iterative methods. *SIAM J. Matrix Anal. Appl.* 4(1):111–115.

Von zur Gathen J, Gerhard J (2003) *Modern Computer Algebra* (Cambridge University Press, Cambridge, UK).

Wan Z (2006) An algorithm to solve integer linear systems exactly using numerical methods. *J. Symbolic Comput.* 41(6):621–632.

Wilkinson JH (1963) *Rounding Errors in Algebraic Processes* (Prentice Hall, Englewood Cliffs, NJ).

Wunderling R (1996) Paralleler und objektorientierter simplex-algorithmus. Unpublished doctoral thesis, Technische Universität Berlin.

Yap CK (1997) Robust geometric computation. Goodman JE, O'Rourke J, eds. *Handbook of Discrete and Computational Geometry* (CRC Press, Boca Raton, FL), 653–668.