Mathematical Programming Computation (2020) 12:603–635
https://doi.org/10.1007/s12532-020-00181-8

FULL LENGTH PAPER

Implementation of an interior point method with basis preconditioning

Lukas Schork[1] · **Jacek Gondzio**[1]

Received: 21 September 2018 / Accepted: 5 February 2020 / Published online: 24 February 2020

Abstract
The implementation of a linear programming interior point solver is described that is based on iterative linear algebra. The linear systems are preconditioned by a basis matrix, which is updated from one interior point iteration to the next to bound the entries in a certain tableau matrix. The update scheme is based on simplex-type pivot operations and is implemented using linear algebra techniques from the revised simplex method. An initial basis is constructed by a crash procedure after a few interior point iterations. The basis at the end of the interior point solve provides the starting basis for a crossover method which recovers a basic solution to the linear program. Results of a computational study on a diverse set of medium to large-scale problems are discussed.

Keywords Linear programming · Interior point methods · Basis preconditioning · Crossover

Mathematics Subject Classification 90C05 · 90C06 · 90C51

1 Introduction

The purpose of this paper is to describe a new linear programming (LP) interior point solver called IPX that is based on iterative linear algebra. IPX uses an approach originally implemented in [1,19] that employs a basis matrix as preconditioner for the linear systems. In the previous works the basis was chosen as the first m linearly inde-

Supported by Google Research Award "Fast interior point method for linear programming problems".

✉ Lukas Schork
L.Schork@ed.ac.uk

Jacek Gondzio
J.Gondzio@ed.ac.uk

[1] School of Mathematics, University of Edinburgh, Edinburgh EH9 3FD, Scotland, UK

pendent columns of the LP constraint matrix $A \in \mathbb{R}^{m \times n}$, after ordering the columns by taking scaling factors from the current interior point iterate into account. (In [1] the scaling factors were used for ordering, whereas in [19] the 1-norm of the scaled columns was used.) Although this yields a perfect preconditioner close to the solution of a nondegenerate LP model, the success of the method was limited in practice.

The core of IPX is a new method for basis selection. A starting basis is constructed after a few interior point iterations and subsequently updated in an attempt to maintain "maximum volume". The maximum volume criterion is known from rank revealing matrix factorizations (see, for example, [21]) and bounds the condition number of the preconditioned matrices in terms of the dimension of the LP model [3,25]. Because finding a maximum volume basis is computationally impractical for large-scale problems, the implementation is based on a heuristic that efficiently finds a basis of comparable quality. The required linear algebra operations are those from the revised simplex method, for which established and highly optimized computational techniques exist.

Section 2 sketches the interior point algorithm and discusses the effect of inexact step directions, which arise from iterative linear algebra. Section 3 presents the basis preconditioner and the update scheme for maintaining a basis matrix. Section 4 is concerned with the very first interior point iterations and the construction of a starting basis. Section 5 presents the crossover method for recovering a vertex solution from the final basis. Computational results on a representative set of LP models are discussed in Sect. 6. This test set is used throughout the paper to provide statistics of IPX runs and to illustrate the effect of algorithmic decisions.

The following notations and conventions are used. When $\mathbf{d}$ is a vector, D is the diagonal matrix with entries d_j on the diagonal. Index sets are understood to be ordered and are denoted by calligraphic letters; $\mathcal{J}_k$ is the kth index and $|\mathcal{J}|$ the number of indices in the set. When D is a diagonal matrix, $D_{\mathcal{B}}$ is the principal submatrix indexed by $\mathcal{B}$. For the rectangular matrix A, $A_{\mathcal{B}}$ is composed of the corresponding columns of A. A basis $\mathcal{B}$ is an index set such that $A_{\mathcal{B}}$ is square and nonsingular. Associated with $\mathcal{B}$ is the nonbasic set $\mathcal{N}$ with the obvious definition.

IPX internally stores the LP model in the following primal-dual form

$$\begin{aligned} \underset{\mathbf{x},\mathbf{x}^l,\mathbf{x}^u}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} - \mathbf{x}^l = \mathbf{l}, \\ & \mathbf{x} + \mathbf{x}^u = \mathbf{u}, \\ & \mathbf{x}^l, \mathbf{x}^u \geq \mathbf{0}, \end{aligned} \tag{1a}$$

and

$$\begin{aligned} \underset{\mathbf{y},\mathbf{z}^l,\mathbf{z}^u}{\text{maximize}} \quad & \mathbf{b}^T\mathbf{y} + \mathbf{l}^T\mathbf{z}^l - \mathbf{u}^T\mathbf{z}^u \\ \text{subject to} \quad & A^T\mathbf{y} + \mathbf{z}^l - \mathbf{z}^u = \mathbf{c}, \\ & \mathbf{z}^l, \mathbf{z}^u \geq \mathbf{0}, \end{aligned} \tag{1b}$$

where A is an $m \times (n+m)$ matrix whose rightmost m columns form the identity matrix. The first n and the last m components of $\mathbf{x}$ are termed "structural" and "slack" variables, respectively, although the objective coefficients of slack variables do not need to be zero. Entries of $-\mathbf{l}$ and $\mathbf{u}$ are allowed to be infinity, in which case the corresponding dual variable is fixed at zero and its term is dropped from the dual objective. We define the index sets

$$\mathscr{L} = \{j \mid l_j > -\infty\}, \quad \mathscr{U} = \{j \mid u_j < \infty\}.$$

A variable x_j is termed "free" if it has no finite bound and "fixed" if its lower and upper bound are equal. After preprocessing only structural variables can be free and only slack variables can be fixed.

IPX accepts user input in the more convenient form

$$\begin{aligned} \underset{\bar{\mathbf{x}}}{\text{minimize}} \quad & \bar{\mathbf{c}}^T \bar{\mathbf{x}} \\ \text{subject to} \quad & \bar{A}\bar{\mathbf{x}} \{=, \leq, \geq\} \bar{\mathbf{b}}, && \text{(2a)} \\ & \bar{\mathbf{l}} \leq \bar{\mathbf{x}} \leq \bar{\mathbf{u}}, && \text{(2b)} \end{aligned}$$

where $\bar{A}$ is an $\bar{m} \times \bar{n}$ matrix. The user model is transformed into computational form with optional dualization. If the input becomes the primal problem, then slack variables $\mathbf{s} \in \mathbb{R}^{\bar{m}}$ with bounds

$$\begin{aligned} 0 \leq s_i \leq 0 \quad & \text{if constraint } i \text{ is of type " = "} \\ 0 \leq s_i \leq \infty \quad & \text{if constraint } i \text{ is of type " } \leq \text{ "} \\ -\infty \leq s_i \leq 0 \quad & \text{if constraint } i \text{ is of type " } \geq \text{ "} \end{aligned}$$

are introduced to transform the general constraints (2a) into

$$\bar{A}\bar{\mathbf{x}} + I_{\bar{m}}\mathbf{s} = \bar{\mathbf{b}},$$

where $I_{\bar{m}}$ denotes the identity matrix of dimension $\bar{m}$. Defining $\mathbf{x}^T = \left(\bar{\mathbf{x}}^T, \mathbf{s}^T\right)$, we obtain (1a) with the constraint matrix $A = \left[\bar{A}\ I_{\bar{m}}\right]$ in the desired form.

If the input problem is dualized, then in a first step variables that have infinite lower bound but finite upper bound are negated. The remaining finite upper bounds are treated as general constraints of the form (2a), so that all variables have lower bounds only (which can be $-\infty$). After these transformations, standard LP dualization rules can be applied to derive the dual of the input problem, which is then put into computational form as above. The constraint matrix becomes $A = \left[\bar{A}^T\ (-I_{\bar{n}})_{\mathscr{J}}\ I_{\bar{n}}\right]$, where $\mathscr{J}$ is the set of variables with finite lower and upper bounds. The reason for dualization is that the linear algebra implementation is optimized for matrices A with (many) more structural columns than rows. If the user does not make an explicit choice, the model is dualized if $\bar{m} > 2\bar{n}$.

2 Inexact interior point method

A primal-dual interior point method (IPM) simultaneously solves (1a) and (1b) by generating a sequence of iterates for $(\mathbf{x}, \mathbf{x}^l, \mathbf{x}^u, \mathbf{y}, \mathbf{z}^l, \mathbf{z}^u)$ that keeps $(\mathbf{x}^l, \mathbf{x}^u, \mathbf{z}^l_{\mathscr{L}}, \mathbf{z}^u_{\mathscr{U}})$ positive while driving the residuals

$$\mathbf{r}^b = \mathbf{b} - A\mathbf{x}, \tag{3a}$$
$$\mathbf{r}^l = \mathbf{l} - \mathbf{x} + \mathbf{x}^l, \tag{3b}$$
$$\mathbf{r}^u = \mathbf{u} - \mathbf{x} - \mathbf{x}^u, \tag{3c}$$
$$\mathbf{r}^c = \mathbf{c} - A^T\mathbf{y} - \mathbf{z}^l + \mathbf{z}^u \tag{3d}$$

and the complementarity measure

$$\mu = \frac{1}{n_\mu}\left(\sum_{j\in\mathscr{L}} x^l_j z^l_j + \sum_{j\in\mathscr{U}} x^u_j z^u_j\right)$$

to zero. Here $n_\mu = |\mathscr{L}| + |\mathscr{U}|$ is the number of finite bounds. If the iterate is feasible, i. e. (3a)–(3d) are all zero, then $n_\mu\mu = f_p - f_d$ is the gap between the primal and dual objective values. The usual requirement is to obtain 8-digit accuracy in the objective, leading to the termination criterion

$$|f_p - f_d| \le 10^{-8}\left(1 + 0.5|f_p + f_d|\right). \tag{4}$$

By the behaviour of the algorithm, primal and dual feasibility are practically always satisfied when the objective criterion is reached.

The most time consuming part of each interior point iteration is computing the step direction by solving a small number of linear systems of the form

$$\begin{bmatrix} & A^T & & & I & -I \\ A & & & & & \\ I & & -I & & & \\ I & & & I & & \\ & & Z^l & & X^l & \\ & & & Z^u & & X^u \end{bmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \\ \Delta\mathbf{x}^l \\ \Delta\mathbf{x}^u \\ \Delta\mathbf{z}^l \\ \Delta\mathbf{z}^u \end{pmatrix} = \begin{pmatrix} \mathbf{r}^c \\ \mathbf{r}^b \\ \mathbf{r}^l \\ \mathbf{r}^u \\ \mathbf{s}^l \\ \mathbf{s}^u \end{pmatrix} \tag{5}$$

for certain vectors $\mathbf{s}^l$ and $\mathbf{s}^u$. IPX implements a version of Mehrotra's method [17] that composes the step from a predictor and a corrector direction. Mehrotra's method is widely considered to be the most efficient method using two linear system solves per iteration. For implementations based on Cholesky factorization it is common practice to compute more than one corrector direction to reduce the iteration count of the IPM [8]. We do not consider this technique here because iterative linear algebra does not offer large savings when solving multiple systems with the same matrix.

After eliminating $\Delta\mathbf{x}^l$, $\Delta\mathbf{x}^u$, $\Delta\mathbf{z}^l$ and $\Delta\mathbf{z}^u$ from (5), the linear system becomes

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ -\Delta\mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{r}^a \\ \mathbf{r}^b \end{pmatrix} \tag{6}$$

for a certain vector $\mathbf{r}^a$ and G the diagonal matrix with entries $g_j = z_j^l/x_j^l + z_j^u/x_j^u$. Note that g_j is zero if x_j is a free variable. To derive accuracy conditions that an approximate solution computed by an iterative method must satisfy, it is useful to define diagonal matrices D and H with entries

$$d_j = \begin{cases} (g_j)^{-1/2} & \text{if } g_j \neq 0, \\ 1 & \text{if } g_j = 0, \end{cases} \qquad h_j = \begin{cases} 1 & \text{if } g_j \neq 0, \\ 0 & \text{if } g_j = 0, \end{cases}$$

and to rescale (6) to

$$\begin{bmatrix} H & DA^T \\ AD & 0 \end{bmatrix} \begin{pmatrix} D^{-1}\Delta\mathbf{x} \\ -\Delta\mathbf{y} \end{pmatrix} = \begin{pmatrix} D\mathbf{r}^a \\ \mathbf{r}^b \end{pmatrix}. \tag{7}$$

The d_j are called "scaling factors" in interior point terminology and the 2×2 matrices are said to have "KKT form". The KKT system can either be solved as it is written or can be further reduced by eliminating (parts of) $\Delta\mathbf{x}$. In any case, an iterative linear solver only provides an approximate solution to (6) or (7) and the accuracy of the solution needs to be controlled by some means. The iterative methods implemented in IPX compute a solution of the form

$$\begin{bmatrix} H & DA^T \\ AD & 0 \end{bmatrix} \begin{pmatrix} D^{-1}\Delta\mathbf{x} \\ -\Delta\mathbf{y} \end{pmatrix} = \begin{pmatrix} D\mathbf{r}^a \\ \mathbf{r}^b \end{pmatrix} + \begin{pmatrix} \mathbf{t} \\ \mathbf{0} \end{pmatrix},$$

where a residual $\mathbf{t}$ occurs only in the first block equation. As termination criterion for the linear solver it is required that

$$\|\mathbf{t}\|_\infty \leq \tau := \varepsilon_\tau \sqrt{\mu}, \tag{8}$$

where ε_τ is a problem independent parameter. The criterion is motivated theoretically because it maintains boundedness of $\left\|D^{-1}\Delta\mathbf{x}\right\|$ in terms of $\sqrt{\mu}$, which is key to the convergence of the IPM (see, for example, [27, Chapter 6]). Choosing a smaller tolerance increases the computational cost per linear system but reduces the number of interior point iterations as the step directions become more accurate. The effect on the total computation time is quite moderate, however. On our test set, varying ε_τ between its default value 0.3 and within the range [0.05, 0.5] changed the geometric mean of the IPX runtimes by up to 8%.

Given an approximate solution to the KKT system, the order in which the remaining step components are recovered determines to which of the block equations in (5) the residual propagates. Solving the dual feasibility equations exactly has shown to be necessary for the robustness of the IPM. We therefore compute

$$
\begin{aligned}
&\Delta \mathbf{x}^l = -\mathbf{r}^l + \Delta \mathbf{x}, \\
&\Delta \mathbf{x}^u = \mathbf{r}^u - \Delta \mathbf{x}, \\
&\Delta z_j^l = (s_j^l - z_j^l \Delta x_j^l)/x_j^l && \text{for all } j \text{ for which } z_j^l/x_j^l < z_j^u/x_j^u, \\
&\Delta z_j^u = (s_j^u - z_j^u \Delta x_j^u)/x_j^u && \text{for all } j \text{ for which } z_j^l/x_j^l \geq z_j^u/x_j^u, \\
&\Delta z_j^l = r_j^c - (A^T \Delta \mathbf{y})_j + \Delta z_j^u && \text{for all } j \text{ for which } z_j^l/x_j^l \geq z_j^u/x_j^u, \\
&\Delta z_j^u = -r_j^c + (A^T \Delta \mathbf{y})_j + \Delta z_j^l && \text{for all } j \text{ for which } z_j^l/x_j^l < z_j^u/x_j^u.
\end{aligned}
$$

By this choice the first four block equations in (5) are solved exactly and the residual is shifted between the last two blocks. If a variable has two finite bounds and is active at its lower bound in the solution, the residual eventually occurs in the equation $z_j^l \Delta x_j^l + x_j^l \Delta z_j^l = s_j^l$. Since in this case $x_j^l \to 0$ but $x_j^u \to (x_j^u)^* > 0$, adjusting Δz_j^l to satisfy dual feasibility causes a smaller residual in the right-hand side to (5) than adjusting Δz_j^u.

3 Basis preconditioning

In advanced interior point iterations the KKT matrix usually becomes very ill conditioned due to a wide spread of the diagonal entries of G. This ill conditioning is a major obstacle for iterative linear algebra and requires effective preconditioning to obtain an acceptable iteration count of the linear solver. The situation is further complicated because LP problems arise from a variety of applications and therefore have different structural and numerical properties. The motivation for IPX was to develop a general-purpose LP solver based on iterative linear algebra, which is not tailored to a particular class of problems.

We shall see that preconditioning the linear systems by a careful choice of basis matrix yields a robust method with good overall efficiency. Different variants of basis preconditioning were described in [1,3,19]. In this paper a new variant specific for IPMs is presented that takes advantage of free variables while reducing the KKT system to a smaller, positive definite one.

3.1 Preconditioned CR method

Assume first that the LP model does not contain free variables. Then G has a zero-free diagonal, $D = G^{-1/2}$ and the KKT system (6) can be reduced to positive definite normal equations

$$AD^2A^T \Delta \mathbf{y} = \mathbf{r}^b - AD^2\mathbf{r}^a =: \mathbf{r}.$$

Basis preconditioning transforms this system using a scaled basis matrix $A_{\mathscr{B}}D_{\mathscr{B}}$ into

$$(A_{\mathscr{B}}D_{\mathscr{B}})^{-1} AD^2A^T (A_{\mathscr{B}}D_{\mathscr{B}})^{-T} \Delta \mathbf{u} = (A_{\mathscr{B}}D_{\mathscr{B}})^{-1}\mathbf{r}. \tag{9}$$

In our setting the transformation is carried out explicitly, meaning that the iterative solver computes a solution for $\Delta \mathbf{u}$ rather than $\Delta \mathbf{y}$.

IPX uses the Conjugate Residual (CR) method [23, Algorithm 6.20] for solving positive definite linear systems. The CR method is a Krylov subspace method, orig-

inally proposed in [13], whose iterates minimize the 2-norm of the residual vector. CR is equivalent to the minimum residual (MINRES) method [20] when the latter is applied to a positive definite system, see [7]. Compared to the Conjugate Gradient method, CR requires one more vector update per iteration, but in our setting has shown to reach the termination criterion in fewer iterations and smaller computation time.

Matrix-vector products with the transformed normal matrix

$$C := (A_{\mathscr{B}} D_{\mathscr{B}})^{-1} A D^2 A^T (A_{\mathscr{B}} D_{\mathscr{B}})^{-T} = I_m + (A_{\mathscr{B}} D_{\mathscr{B}})^{-1} A_{\mathscr{N}} D^2_{\mathscr{N}} A^T_{\mathscr{N}} (A_{\mathscr{B}} D_{\mathscr{B}})^{-T} \tag{10}$$

are implemented through a matrix-vector product with $A_{\mathscr{N}} D^2_{\mathscr{N}} A^T_{\mathscr{N}}$ and two linear system solves with a factorization of $A_{\mathscr{B}} D_{\mathscr{B}}$. Neither $A_{\mathscr{N}} D^2_{\mathscr{N}} A^T_{\mathscr{N}}$ nor C are formed explicitly.

After computing an approximate solution for $\Delta\mathbf{u}$, a solution to the KKT system is recovered from

$$\begin{aligned}
\Delta\mathbf{y} &= (A_{\mathscr{B}} D_{\mathscr{B}})^{-T} \Delta\mathbf{u}, \\
\Delta\mathbf{x}_{\mathscr{N}} &= D^2_{\mathscr{N}} (\mathbf{r}^a_{\mathscr{N}} + A^T_{\mathscr{N}} \Delta\mathbf{y}), \\
\Delta\mathbf{x}_{\mathscr{B}} &= A^{-1}_{\mathscr{B}} (\mathbf{r}^b - A_{\mathscr{N}} \Delta\mathbf{x}_{\mathscr{N}}).
\end{aligned}$$

By definition of $\Delta\mathbf{x}$, a residual originating from the iterative method occurs only in the basic components of the first block equation in (7). That residual is

$$\begin{aligned}
\mathbf{t}_{\mathscr{B}} &= D^{-1}_{\mathscr{B}} \Delta\mathbf{x}_{\mathscr{B}} - D_{\mathscr{B}} A^T_{\mathscr{B}} \Delta\mathbf{y} - D_{\mathscr{B}} \mathbf{r}^a_{\mathscr{B}} \\
&= D^{-1}_{\mathscr{B}} A^{-1}_{\mathscr{B}} (\mathbf{r}^b - A_{\mathscr{N}} \Delta\mathbf{x}_{\mathscr{N}}) - \Delta\mathbf{u} - D_{\mathscr{B}} \mathbf{r}^a_{\mathscr{B}} \\
&= D^{-1}_{\mathscr{B}} A^{-1}_{\mathscr{B}} \mathbf{r}^b - D^{-1}_{\mathscr{B}} A^{-1}_{\mathscr{B}} A_{\mathscr{N}} D^2_{\mathscr{N}} (\mathbf{r}^a_{\mathscr{N}} + A^T_{\mathscr{N}} \Delta\mathbf{y}) - \Delta\mathbf{u} - D_{\mathscr{B}} \mathbf{r}^a_{\mathscr{B}} \\
&= (A_{\mathscr{B}} D_{\mathscr{B}})^{-1} \mathbf{r}^b - (A_{\mathscr{B}} D_{\mathscr{B}})^{-1} A D^2 \mathbf{r}^a \\
&\quad - (I_m + (A_{\mathscr{B}} D_{\mathscr{B}})^{-1} A_{\mathscr{N}} D^2_{\mathscr{N}} A^T_{\mathscr{N}} (A_{\mathscr{B}} D_{\mathscr{B}})^{-T}) \Delta\mathbf{u} \\
&= (A_{\mathscr{B}} D_{\mathscr{B}})^{-1} \mathbf{r} - C \Delta\mathbf{u}.
\end{aligned}$$

Therefore, to satisfy the accuracy requirement (8), the CR method is terminated when the infinity norm of the residual in (9) becomes smaller than τ.

When G has zeros on the diagonal, the KKT system can still be reduced to normal equations if all free variables are basic. This requirement is satisfied by the initial basis in IPX and is maintained throughout; hence let $\mathscr{B} = \mathscr{B}_0 \cup \mathscr{B}_1$, where $\mathscr{B}_0$ are the indices of all free variables. Then (7) can be permuted to

$$\begin{bmatrix}
I_{\mathscr{N}} & & D_{\mathscr{N}} A^T_{\mathscr{N}} \\
& \begin{bmatrix} I_{\mathscr{B}_1} & \\ & 0 \end{bmatrix} & D_{\mathscr{B}} A^T_{\mathscr{B}} \\
A_{\mathscr{N}} D_{\mathscr{N}} & A_{\mathscr{B}} D_{\mathscr{B}} & 0
\end{bmatrix}
\begin{pmatrix} D^{-1}_{\mathscr{N}} \Delta\mathbf{x}_{\mathscr{N}} \\ D^{-1}_{\mathscr{B}} \Delta\mathbf{x}_{\mathscr{B}} \\ -\Delta\mathbf{y} \end{pmatrix}
=
\begin{pmatrix} D_{\mathscr{N}} \mathbf{r}^a_{\mathscr{N}} \\ D_{\mathscr{B}} \mathbf{r}^a_{\mathscr{B}} \\ \mathbf{r}^b \end{pmatrix}.$$

Transforming this system using the scaled basis matrix yields

$$\begin{bmatrix} I_{\mathcal{N}} & & D_{\mathcal{N}} A_{\mathcal{N}}^T A_{\mathcal{B}}^{-T} D_{\mathcal{B}}^{-1} \\ & \begin{bmatrix} I_{\mathcal{B}_1} & 0 \end{bmatrix} & I_{\mathcal{B}} \\ D_{\mathcal{B}}^{-1} A_{\mathcal{B}}^{-1} A_{\mathcal{N}} D_{\mathcal{N}} & I_{\mathcal{B}} & 0 \end{bmatrix} \begin{pmatrix} D_{\mathcal{N}}^{-1} \Delta \mathbf{x}_{\mathcal{N}} \\ D_{\mathcal{B}}^{-1} \Delta \mathbf{x}_{\mathcal{B}} \\ -\Delta \mathbf{u} \end{pmatrix} = \begin{pmatrix} D_{\mathcal{N}} \mathbf{r}_{\mathcal{N}}^a \\ D_{\mathcal{B}} \mathbf{r}_{\mathcal{B}}^a \\ D_{\mathcal{B}}^{-1} A_{\mathcal{B}}^{-1} \mathbf{r}^b \end{pmatrix},$$

where $\Delta \mathbf{u} = D_{\mathcal{B}} A_{\mathcal{B}}^T \Delta \mathbf{y}$ as before. Notice that the components of $\Delta \mathbf{u}$ correspond to basic variables. It follows from the second block equation and $D_{\mathcal{B}_0}$ being the identity matrix that the components of $-\Delta \mathbf{u}$ corresponding to free variables are given by $\mathbf{r}_{\mathcal{B}_0}^a$ and can be eliminated. Also, the components of $\Delta \mathbf{x}_{\mathcal{B}}$ corresponding to free variables can be removed from the linear system and can be computed from the third block equation once the remaining components of $D^{-1} \Delta \mathbf{x}$ are known. Let $P = \begin{bmatrix} I_{\mathcal{B}_1} & 0 \end{bmatrix}$ have m columns and $S = P D_{\mathcal{B}}^{-1} A_{\mathcal{B}}^{-1} A_{\mathcal{N}} D_{\mathcal{N}}$. The resulting linear system is

$$\begin{bmatrix} I_{\mathcal{N}} & & S^T \\ & I_{\mathcal{B}_1} & I_{\mathcal{B}_1} \\ S & I_{\mathcal{B}_1} & 0 \end{bmatrix} \begin{pmatrix} D_{\mathcal{N}}^{-1} \Delta \mathbf{x}_{\mathcal{N}} \\ D_{\mathcal{B}_1}^{-1} \Delta \mathbf{x}_{\mathcal{B}_1} \\ -P \Delta \mathbf{u} \end{pmatrix} = \begin{pmatrix} D_{\mathcal{N}} \mathbf{r}_{\mathcal{N}}^a \\ D_{\mathcal{B}_1} \mathbf{r}_{\mathcal{B}_1}^a \\ P D_{\mathcal{B}}^{-1} A_{\mathcal{B}}^{-1} \mathbf{r}^b \end{pmatrix} - \begin{pmatrix} \mathbf{w} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix},$$

where

$$\mathbf{w} = D_{\mathcal{N}} A_{\mathcal{N}}^T A_{\mathcal{B}}^{-T} D_{\mathcal{B}}^{-1} \begin{pmatrix} \mathbf{0} \\ \mathbf{r}_{\mathcal{B}_0}^a \end{pmatrix} = D_{\mathcal{N}} A_{\mathcal{N}}^T A_{\mathcal{B}}^{-T} \begin{pmatrix} \mathbf{0} \\ \mathbf{r}_{\mathcal{B}_0}^a \end{pmatrix}.$$

Because the upper left 2×2 block now has a zero-free diagonal, the system can be reduced to normal equations

$$(I_{\mathcal{B}_1} + S S^T)(P \Delta \mathbf{u}) = -S(D_{\mathcal{N}} \mathbf{r}_{\mathcal{N}}^a - \mathbf{w}) - D_{\mathcal{B}_1} \mathbf{r}_{\mathcal{B}_1}^a + P D_{\mathcal{B}}^{-1} A_{\mathcal{B}}^{-1} \mathbf{r}^b \tag{11}$$

and can be solved for $P \Delta \mathbf{u}$. The solution for $\Delta \mathbf{u}$ is completed by inserting the components corresponding to free variables given by $-\mathbf{r}_{\mathcal{B}_0}^a$.

The dimension of (11) is m minus the number of free variables. In our implementation the CR method always operates on vectors of dimension m, in which components corresponding to free variables are initialized to zero and are reset to zero after each matrix-vector product with C defined in (10). The only actual change to the code was to replace the right-hand side of (9) by that of (11), scattered into a full size vector. The result is a concise handling of free variables, which implicitly reduces the dimension of the linear systems.

3.2 Maintaining a basis matrix

The aim of basis preconditioning is to make the transformed normal matrix (10) well conditioned, since then we expect fast convergence of the linear solver. Clearly, the condition number of C depends on the choice of the basis. While finding an optimal basis in that sense seems to be impossible without enumerating all bases [3], a criterion

that can be targeted in practice is to find a basis $\mathscr{B}$ that satisfies[1]

$$\max_{p,q} |D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_{\mathscr{N}} D_{\mathscr{N}}|_{pq} \leq \rho \tag{12}$$

for a parameter $\rho \geq 1$. Such a basis, called "ρ-maximum volume basis", exists for any matrix AD of full row rank and bounds the spectrum of C in the interval $[1, 1+\rho^2 mn]$, see [3,25].

Compared to the previous approaches for basis preconditioning in IPMs [1,19], the advantage of the maximum volume concept is that it does not rely on having m large and n small scaling factors. Due to degeneracy, the latter assumption is rarely satisfied in practice. Furthermore, the bound on the spectrum of C stated above gives reason to expect that the preconditioner is not only effective asymptotically, but also in earlier interior point iterations.

A ρ-maximum volume basis is obtained through pivot operations in the scaled tableau matrix $D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_{\mathscr{N}} D_{\mathscr{N}}$ starting from an arbitrary basis [9,15]. A generic pivot scheme is given in Algorithm 1. In each iteration any entry that is larger than ρ in absolute value can be chosen as pivot. Because each basis update increases $|\det(A_{\mathscr{B}} D_{\mathscr{B}})|$ (see [9]), the algorithm terminates in a finite number of iterations.

Algorithm 1 Maxvolume

Input: basis $\mathscr{B}$, parameter $\rho \geq 1$.
1: **loop**
2: Choose (p, q) such that $|D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_{\mathscr{N}} D_{\mathscr{N}}|_{pq} > \rho$. If no such (p, q) exists, then stop.
3: $\mathscr{B}_p = \mathscr{N}_q$
4: **end loop**

The numerical experiments in Sect. 3.3 will show that the number of pivot operations for updating a maximum volume basis from one interior point iteration to the next is quite small in practice. The difficulty in implementing Algorithm 1 is finding a pivot element in line 2. Because the tableau matrix is only available implicitly through operations with $A_{\mathscr{B}}^{-1}$, searching systematically for an entry that is larger than ρ can be expensive. In particular, testing if (12) is satisfied would require to compute all tableau entries, which is impractical for large-scale problems.

To make the approach practical we relax the target and only *attempt* to find a ρ-maximum volume basis. The idea is to perform pivot operations in the scaled tableau matrix as long as large entries are readily found, and to terminate the method when the pivot search becomes costly. The pseudocode of the update procedure in IPX is given in Algorithm 2.

Assume first that the parameter *nslices* is 1. Then w_j is the sum of entries in column j of the scaled tableau matrix if $j \in \mathscr{N}$. The algorithm chooses the index with maximum such weight as candidate and computes the corresponding column of the scaled tableau matrix. If its maximum entry is larger than ρ in absolute value, the basis is updated. In this case the tableau row needs to be computed as well to update

[1] For a matrix A the expression $|A|$ is meant componentwise.

Algorithm 2 Maxvolume Heuristic

Input: basis $\mathscr{B}$, parameters $(\rho, nslices, maxskip)$ such that $\rho \geq 1$, $1 \leq nslices \leq m$, $maxskip \geq 0$.

1: **for** $s = 0$ to $nslices - 1$ **do**
2: Let $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{w} \in \mathbb{R}^{n+m}$.
3: **for** $i = 1$ to m **do** // set row mask
4: $u_i = 1$ if $\text{mod}(i, nslices) = s$
5: $u_i = 0$ if $\text{mod}(i, nslices) \neq s$
6: **end for**
7: $\mathbf{w}_{\mathscr{N}}^T = \mathbf{u}^T D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_{\mathscr{N}} D_{\mathscr{N}}$, $\mathbf{w}_{\mathscr{B}} = \mathbf{0}$ // initialize column weights
8: $nskipped = 0$ // count "skipped" columns
9: **while** $nskipped \leq maxskip$ **do**
10: $j = \arg\max_k |w_k|$ // choose candidate column
11: Compute $\mathbf{v} = D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_j d_j$.
12: **if** $\|\mathbf{v}\|_\infty > \rho$ **then**
13: $p = \arg\max_i |v_i|$
14: $\mathscr{B}_p = j$
15: Update $\mathbf{w}$.
16: **else** // no pivot found in column j
17: $nskipped = nskipped + 1$
18: Flag column j to be excluded from pivot search.
19: **end if**
20: **end while**
21: **end for**

the column sums (line 15). After computing $maxskip + 1$ candidate columns without finding a pivot, the algorithm terminates.

When $nslices > 1$, then in each iteration of the outer loop only a subset of rows of the tableau matrix contributes to the column weights. This slicing of the tableau matrix decreases the chance that when a column has large positive and negative entries that these cancel out in the column sum; and it makes the algorithm adaptive to the number of rows of A.

IPX runs Algorithm Maxvolume Heuristic at the beginning of each interior point iteration with the new scaling matrix D and the basis from the previous iteration as starting basis. Free and fixed variables are excluded from the pivot search and remain basic and nonbasic, respectively. The default parameters are $nslices = 5 + \lfloor m/10000 \rfloor$ and $maxskip = 10$, but they have shown little effect on the number of basis updates and the quality of the basis preconditioner.

3.3 Empirical tests

To investigate the effect of basis selection on the number of CR iterations, we consider two LP models, `pds-20` and `nug20`, which have similar dimensions but very different characteristics. IPX was run on these models twice, once using Algorithm Maxvolume Heuristic for updating the basis, and once using Algorithm Maxvolume, which yields a (true) ρ-maximum volume basis. Algorithm Maxvolume was implemented by sequentially computing columns of $D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_{\mathscr{N}} D_{\mathscr{N}}$ and updating the basis when the maximum absolute entry of the column was larger than ρ. The algorithm terminated after having computed all columns without requiring an update. We note that this was a very expensive procedure for `nug20` due to a large number of

Table 1 Total number of basis updates and CR iterations, excluding initial IPM iterations

Name (dimension)	ρ	Maxvolume Heuristic		Maxvolume	
		Updates	CR iter	Updates	CR iter
`pds-20`	1.1	2708	3700	3546	3554
($m = 12081, n = 81163$)	2.0	1899	4370	1981	4109
	4.0	1654	5738	1648	5564
	10.0	1492	9225	1493	8869
`nug20`	1.1	15,051	22,028	29,184	17,559
($m = 14{,}098, n = 72{,}546$)	2.0	10,383	25,654	11,993	22,033
	4.0	8455	34,299	9075	29,854
	10.0	7137	58,039	7396	49,747

column computations which did not require a basis update, and is not an option in practice. Both update methods were tested with parameter $\rho \in \{1.1, 2.0, 4.0, 10.0\}$, giving a total of 8 IPM solves per model. The initial IPM iterations and the starting basis (see Sect. 4) were the same for all runs on the same model. No crossover was used.

The total number of basis updates and CR iterations are given in Table 1. Regarding the number of basis updates, there is little difference between Algorithm Maxvolume Heuristic and Algorithm Maxvolume, except for $\rho = 1.1$. This means that if there would exist an efficient method for finding pivot elements, then maintaining a ρ-maximum volume basis for $\rho \geq 2$, say, would be computationally feasible. However, there would be little benefit from this, since for $\rho \geq 2$ the number of CR iterations obtained with the heuristically found basis was at most 17% higher than the number obtained with the maximum volume basis.

The number of CR iterations per interior point iteration (without the initial IPM iterations) are plotted in Fig. 1. It is seen that for both models and all values of ρ the curves closely resemble each other, showing that the heuristically found basis yields an equally good preconditioner as a maximum volume basis. The curves also illustrate the typical behaviour of the CR iteration counts during the course of the IPM: one or more peaks usually occur after the switch to basis preconditioning, and the iterations level out toward the end of the interior point solve.

On the two example models, a smaller ρ systematically reduced the number of CR iterations. This need not always be the case, however. It turned out that sparsity in the tableau matrix also has large effect on the number of CR iterations. While A is almost always a sparse matrix in real life, for some models $A_{\mathscr{B}}^{-1}$ and $A_{\mathscr{B}}^{-1}A_{\mathscr{N}}$ are sparse as well for all relevant bases; such models are called "hypersparse" in simplex community [4,11]. On the one hand hypersparsity enables orders of magnitude speedup for finding the basis because operations with $A_{\mathscr{B}}^{-1}$, if implemented properly, take far less than order of m time. On the other hand a sparse tableau matrix has shown to lead to fewer CR iterations. This is not surprising because the maximum eigenvalue of C is bounded by $1 + \left\| D_{\mathscr{B}}^{-1} A_{\mathscr{B}}^{-1} A_{\mathscr{N}} D_{\mathscr{N}} \right\|_F^2$ (see [25]) and the Frobenius norm becomes small when the

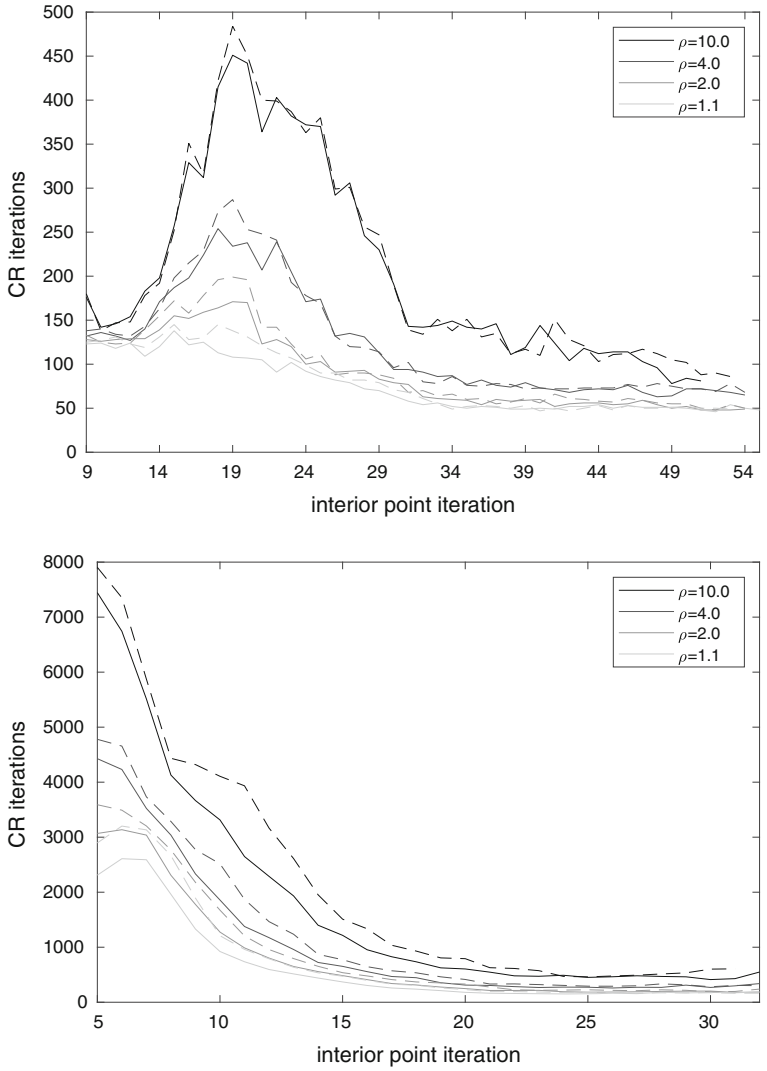

Fig. 1 LP models `pds-20` (above) and `nug20` (below) with ρ-maximum volume basis (solid line) and heuristically computed basis (dashed line)

tableau is sparse. The effect is seen quite impressively by comparing the hypersparse `pds-20` to the not-hypersparse `nug20`. Despite their similar dimensions, the average number of CR iterations per linear system (not shown) was about 10 times higher for `nug20`. When the density of the tableau matrix varies significantly between different bases, it can happen that a smaller ρ leads to more CR iterations. We have observed such a behaviour on a number of LP models. It is not obvious how to choose a basis that preserves sparsity as far as possible while targeting the maximum volume criterion. In the LP context this remains an important topic for further investigation.

When the correlations are as clear as in Table 1, then the optimal ρ in terms of total computation time must trade-off the cost for updating the basis to the reduction of the time spent on the CR method. A larger set of models has shown that values between 2.0 and 4.0 lead to similar and close-to-optimal total runtime. Therefore IPX uses a problem independent default of $\rho = 2.0$.

4 Initial iterations and crash basis

IPX does not use basis preconditioning from the beginning of the interior point solve. At the early iterations the KKT matrix is usually well conditioned by itself and a simpler (and less expensive) preconditioner is equally effective. This is exploited by performing a few "initial iterations" of the IPM in which the CR method is applied with a variant of diagonal scaling. When the method does not converge within $\min\{500, 10 + m/20\}$ iterations, a starting basis is constructed and the interior point iteration is repeated with basis preconditioning. This section looks at these techniques in more detail.

4.1 Initial iterations

The KKT matrix in (6) is nonsingular if and only if A has full row rank and the columns of A corresponding to zeros on the diagonal of G are linearly independent. While the first requirement is satisfied after adding slack variables to all constraints, the latter requirement might not be. To overcome the problem at the beginning, zeros on the diagonal of G are replaced by the regularization value $\min\{\{g_j | \, g_j \neq 0\}, \mu\}$. Regularization can significantly change the solution to the linear system, but using it in the initial iterations has shown to be unproblematic for the convergence of the IPM. The regularization value is chosen to yield a well conditioned KKT matrix as long as the remaining diagonal entries of G are balanced and μ is sufficiently large. Including μ in the definition guarantees that regularization is eventually reduced to zero even if none of the remaining g_j becomes small (i. e. if all non-free variables are at a bound in the solution).

An additional benefit of regularization is that the KKT system can always be reduced to normal equations

$$AG^{-1}A^T \Delta \mathbf{y} = \mathbf{r}^b - AG^{-1}\mathbf{r}^a = \mathbf{r}. \tag{13}$$

From an approximate solution for $\Delta \mathbf{y}$, the solution components $\Delta \mathbf{x}$ to the KKT system are recovered from

$$\Delta \mathbf{x}_{\mathscr{N}} = G_{\mathscr{N}}^{-1}(\mathbf{r}^a + A^T \Delta \mathbf{y})_{\mathscr{N}}, \quad \Delta \mathbf{x}_{\mathscr{B}} = \mathbf{r}^b - A_{\mathscr{N}} \Delta \mathbf{x}_{\mathscr{N}},$$

where G is the regularized matrix and $\mathscr{B} = \{n + 1, \ldots, n + m\}$ is the slack basis. It is easily verified that the accuracy requirement (8) is satisfied if

$$\left\| G_{\mathscr{B}}^{1/2}(\mathbf{r} - AG^{-1}A^T \Delta \mathbf{y}) \right\|_\infty \leq \tau.$$

In the initial iterations IPX solves (13) by the CR method using a symmetric positive definite matrix M as preconditioner. The method iterates on $\Delta\mathbf{y}$ and requires one operation with M^{-1} per iteration. In an early version of the code diagonal preconditioning was used, where $M = \mathrm{diag}(AG^{-1}A^T)$ is obtained by dropping all off-diagonal entries from $AG^{-1}A^T$. The method has been refined for matrices A with "dense" columns; i. e. columns whose nonzero count is much higher than the average. Let A_s and A_d be the sparse and dense part of A, respectively, and G_s and G_d be the corresponding diagonal blocks of G. Then IPX uses

$$M = \mathrm{diag}(A_s G_s^{-1} A_s^T) + A_d G_d^{-1} A_d^T \tag{14}$$

as preconditioner. By treating the second summand as a low-rank update, an inverse representation of M is obtained from the Sherman–Morrison–Woodbury formula and can be computed through the Cholesky factorization of a dense matrix of dimension equal to the number of columns in A_d. For LP models with a small number of dense columns, using (14) is little more expensive than diagonal scaling but often more effective because $A_s G_s^{-1} A_s^T$ is better approximated by its diagonal than $AG^{-1}A^T$. IPX classifies the maximum number of columns as dense such that each column in A_d has more than 40 nonzeros and more than 10 times the number of nonzeros of any column in A_s. If this yields more than 1000 dense columns, then no columns are treated as dense.

4.2 Crash basis

At the switch to basis preconditioning a starting basis must be determined, given the current interior point iterate and its associated scaling matrix D. To reduce the number of basis updates in the first run of Algorithm Maxvolume Heuristic and to make the linear algebra operations fast, the basis should have the following (often competing) properties:

- The basis matrix is well conditioned.
- The basis matrix and ideally the tableau matrix are sparse.
- The basis is close to a maximum volume basis for the current D.
- All free variables are basic.
- All fixed variables are nonbasic.

Making all free variables basic is required for the reduction of the KKT system to normal equations (Sect. 3.1) and is always achievable. If the columns corresponding to free variables are linearly dependent, then the model is either dual infeasible or some of the variables are redundant and can be fixed at an arbitrary value. The analogue requirement is to make all fixed variables nonbasic, which allows them to be removed from the optimization entirely. Again, this is always achievable, for if a fixed slack variable cannot be replaced in the basis, then either is the corresponding row of A redundant and can be removed, or the model is primal infeasible.

Finding efficiently a basis that satisfies the last requirement is already nontrivial if most slack variables are fixed. In particular, the obvious method by computing an LU factorization of the structural part of A would be unacceptably expensive for many

large-scale problems. Instead, IPX "crashes" a starting basis through the following steps:

Initial guess A set $\mathscr{J}$ of m column indices is constructed as follows:

1. If the LP model contains free variables, the first step is computing an incomplete left-looking LU factorization of the corresponding columns of A. In the left-looking method L starts out to be the identity matrix and its strictly lower triangular part is computed one column at a time. Let j be the next free variable, $\hat{\mathbf{a}} = L^{-1}A_j$ and i be such that $|\hat{a}_i|$ is maximum among all entries of $|\hat{\mathbf{a}}|$ whose row has not been pivotal. If $|\hat{a}_i| > 10^{-3}$, variable j is added to $\mathscr{J}$ and the next column of L is composed from the entries of $\hat{\mathbf{a}}/\hat{a}_i$ that are nonzeros in A_j and have not been pivotal. (Restricting the column of L to the nonzero pattern of A_j makes the factorization incomplete.) After processing all free variables, L is discarded but the index set $\mathscr{I}$ of pivot rows is kept.
2. In the second and third step columns of A corresponding to fixed and free variables are treated as vacant; i. e. they are treated as being structurally zero in A and AD. The second step adds singleton columns to $\mathscr{J}$ if their entry in AD is sufficiently large. For each $i \notin \mathscr{I}$ the maximum entry in row i of $|AD|$ and the maximum entry that lies in a singleton column are determined. If the singleton entry is nonzero and not smaller than 0.5 times the maximum of the row, its column is added to $\mathscr{J}$ and i is added to $\mathscr{I}$.
3. Let A_{33} be the submatrix of A composed of rows $i \notin \mathscr{I}$ and columns $j \notin \mathscr{J}$. The third step extends $\mathscr{I}$ and $\mathscr{J}$ by choosing a structurally independent subset of the columns of A_{33}. Processing in decreasing order of the d_j, the next column from A_{33} is tested for being structurally dependent on the columns from A_{33} already chosen, by computing an alternating augmenting path [6]. If the candidate column is independent, its index is added to $\mathscr{J}$ and the row that was newly matched by the augmenting path is added to $\mathscr{I}$. The method stops at the latest when $|\mathscr{J}| = m$ or when all columns of A_{33} have been processed. It is stopped before if too many candidate columns were structurally dependent, ensuring that the computation time is small.
4. Finally $\mathscr{J}$ is completed by adding slack variables for $i \notin \mathscr{I}$.

Initial factorization The task is to find a well conditioned basis matrix that contains as many columns of $\mathscr{J}$ as possible. A right-looking Markowitz LU factorization of $A_{\mathscr{J}}$ is computed with the usual columnwise threshold pivoting. If during the course of the factorization all entries in a column of the active submatrix become smaller than 10^{-3}, the column is immediately removed without choosing a pivot. After completion, the LU factors are padded with unit columns whose row has not been pivotal. Accordingly a preliminary basis $\mathscr{B}$ is obtained composed of indices of $\mathscr{J}$ and indices of slack variables.

Basis repair The resulting basis matrix $A_{\mathscr{B}}$ would be nonsingular in exact arithmetic. It is well known, however, that $A_{\mathscr{B}}$ can have tiny singular values even if no small pivots occur in the LU factorization, and such cases actually happen in practice. It is therefore essential to control the condition number of $A_{\mathscr{B}}$ and to repair numerical singularities if necessary. Because the model is scaled during preprocessing so that the

maximum entry of A is bounded by a moderate number, a high condition number of $A_{\mathscr{B}}$ can only be caused by large entries in $A_{\mathscr{B}}^{-1}$. As described in [14] a rook search is performed for estimating the maximum absolute entry of $A_{\mathscr{B}}^{-1}$ along with its position (p, i) in the matrix. If the entry is larger than 10^5, then $\mathscr{B}_p$ is replaced by the ith slack variable and the rook search is repeated. During the basis repair $A_{\mathscr{B}}$ typically requires frequent refactorization due to numerical instability in the LU update. If the basis matrix is refactorized, columns are dropped from the active submatrix and replaced by unit columns as in the initial factorization.

Handling free and fixed variables The obtained basis may contain fixed slack variables and may not contain all free variables. These "defects" are corrected by basis updates. For each free variable that is nonbasic a column of the tableau matrix is computed and the variable is pivoted into the basis if it can replace a non-free basic variable. If not, the model is either declared dual infeasible or the free variable is fixed at zero. Likewise, for each fixed (slack) variable in the basis a row of the tableau matrix is computed and searched for an exchange column. If the tableau row is numerically zero, the model is either declared primal infeasible or the constraint corresponding to the slack variable is removed. After correcting all defects, the remaining fixed variables (which are nonbasic now) are excluded from the IPM solve.

The described procedure has been developed through extensive testing and works efficiently on most real-world problems. Steps 1–3 of the initial guess are designed to be fast and to yield a column subset of close-to full rank. Depending on the characteristics of the LP model, each of the three steps may add the majority of columns to $\mathscr{J}$. Computing the starting basis typically accounts for less than 5% of the total IPX runtime. When the computation time is significant, the last step of the procedure often dominates due to the number of pivots for fixed variables. Processing these variables as described is advantageous later, however, because otherwise fixed variables can lead to small step sizes in the IPM and dependent equality constraints can cause dual variables to blow up.

The crash bases are surprisingly close to the bases at the end of the interior point solve. In Table 2 the number of basis updates in relation to m is reported. Note that a value 0.25, for example, means that at most 25% of the starting basis have been replaced during the interior point solve. It is seen that maintaining a basis preconditioner requires much fewer than the $2m$–$3m$ basis changes that the simplex method typically needs to solve an LP.

5 Crossover

A basic solution to (1) is an optimal solution with an associated basis $\mathscr{B}$ such that $\mathbf{z}^l_{\mathscr{B}}$ and $\mathbf{z}^u_{\mathscr{B}}$ are zero and $\mathbf{x}_{\mathscr{N}}$ is at a bound (or zero if the variable is free). In this case $\mathscr{B}$ is said to be an optimal LP basis. Basic solutions are required in a number of applications, most prominently for classical integer programming. Because the basis from the preconditioner need not become optimal close to the solution, the basis-preconditioned IPM requires a crossover step for recovering a basic solution, as any other interior point solver.

Table 2 Number of basis updates in the IPM starting from crash basis (165 models in total)

Basis updates/m	Instances
0.000–0.010	24
0.010–0.025	8
0.025–0.050	6
0.050–0.100	16
0.100–0.250	46
0.250–0.500	22
0.500–1.000	38
1.000–3.350	5

For the crossover it is convenient to combine the dual slack variables into $\mathbf{z} = \mathbf{z}^l - \mathbf{z}^u$. A primal-dual solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ then satisfies

$$A\mathbf{x} = \mathbf{b}, \quad A^T\mathbf{y} + \mathbf{z} = \mathbf{c}, \tag{15a}$$
$$\mathbf{l} \le \mathbf{x} \le \mathbf{u}, \tag{15b}$$
$$z_j \le 0 \quad \text{if } x_j > l_j \text{ for all } j, \tag{15c}$$
$$z_j \ge 0 \quad \text{if } x_j < u_j \text{ for all } j. \tag{15d}$$

Let us first assume that $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ from the final interior point iteration is an exact primal-dual solution and let $\mathscr{B}$ be any basis. The variables in $\mathscr{N}^+ = \{j \in \mathscr{N} \mid l_j < x_j < u_j\}$ and $\mathscr{B}^+ = \{j \in \mathscr{B} \mid z_j \neq 0\}$ are called primal and dual "superbasic", respectively. The crossover method removes superbasic variables by two push phases.

The dual push phase manipulates $(\mathbf{y}, \mathbf{z})$ and $\mathscr{B}$ whilst satisfying (15c), (15d) and

$$A_{\mathscr{B}}^T\mathbf{y} + \mathbf{z}_{\mathscr{B}} = \mathbf{c}_{\mathscr{B}},$$
$$A_{\mathscr{N}}^T\mathbf{y} + \mathbf{z}_{\mathscr{N}} = \mathbf{c}_{\mathscr{N}}.$$

In each iteration an $i \in \mathscr{B}^+$ is chosen and z_i is moved toward zero until the adjustment to $(\mathbf{y}, \mathbf{z}_{\mathscr{N}})$ would violate the sign condition for $\mathbf{z}$. The push is complete if z_i reaches zero. Otherwise a basis update is applied exchanging $i \in \mathscr{B}$ with the blocking index $j \in \mathscr{N}$. Because in the latter case z_j has become zero, each iteration reduces the number of dual superbasic variables by 1.

The primal push phase manipulates $\mathbf{x}$ and $\mathscr{B}$ whilst satisfying (15b) and

$$A_{\mathscr{B}}\mathbf{x}_{\mathscr{B}} + A_{\mathscr{N}}\mathbf{x}_{\mathscr{N}} = \mathbf{b}.$$

In each iteration a $j \in \mathscr{N}^+ \cap (\mathscr{L} \cup \mathscr{U})$ is chosen and x_j is moved toward a bound until the adjustment to $\mathbf{x}_{\mathscr{B}}$ would violate the bound constraints. The push is complete if x_j reaches its bound. Otherwise a basis update is applied exchanging $j \in \mathscr{N}$ with the blocking index $i \in \mathscr{B}$. Because in the latter case x_i has been moved to a bound, each iteration reduces the number of primal superbasic variables by 1.

As pointed out in [5], the push phases are equivalent to Megiddo's algorithm [16] if the initial basis contains a maximum number of variables for which $l_j < x_j < u_j$ and a minimum number for which $z_j \neq 0$. In this case the analysis in [16] proves that each push maintains complementarity of $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ so that the final iterate is indeed a basic solution. It is easy to see, however, that each dual push as stated above maintains complementarity regardless of the initial basis; if moving z_i ($i \in \mathscr{B}^+$) toward zero makes a z_j ($j \in \mathscr{N}^+$) nonzero, the step is blocked immediately because (15c) or (15d) would be violated. Furthermore, because there exist no dual superbasic variables after the dual push phase completed, the primal phase can move any basic variable x_i without violating complementarity. Hence the above algorithm can be run from any starting basis.

In practice the final iterate from the IPM is neither exactly feasible nor complementary and care must be taken with small pivot elements to prevent the basis matrix from becoming too ill conditioned for finite precision arithmetic. These issues are addressed by the IPX implementation as follows. Before executing the push phase, the final IPM iterate is dropped to an $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ satisfying (15b)–(15d). For each variable either x_j is set to a bound or z_j is set to zero, depending on which perturbation is smaller. The dropping usually increases the residuals in (15a). Throughout the push operations the conditions (15b)–(15d) are maintained exactly by truncating the update to a variable if necessary. A blocking variable is eligible as pivot only if the pivot element is larger than 10^{-5} in absolute value. Among all candidates, the exchange variable is chosen using the two-pass ratio test from the simplex method [12], which allows larger pivot elements by exploiting a feasibility tolerance for (15b)–(15d) with default value 10^{-7}. The combination of the last two techniques was necessary to make the implementation robust against failure due to a numerically singular basis.

IPX naturally uses the final basis from the preconditioner as starting basis for crossover. The number of pushes is determined from the beginning by the number of superbasic variables, whereas the number of pivot operations depends on the order in which the superbasic variables are processed. IPX proceeds in the primal and dual push phase in decreasing and increasing order, respectively, of the scaling factors from the final interior point iteration. We have not found a systematic difference in the number of pivot operations compared to other orderings and find this the most natural choice that makes the computations invariant to a permutation of the variables.

At the end of the push phases $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a basic solution to the LP model with perturbed right-hand side and objective, and $\mathscr{B}$ may or may not be an optimal basis for the original problem. If it is not, reoptimization with the simplex method is required. The main reason for a non-optimal basis is that the pairwise complementarity products $x_j^l z_j^l$ and $x_j^u z_j^u$ can be quite large when the IPM reaches the termination criterion (4), resulting in a large perturbation. To reduce the number of simplex iterations in the clean-up, IPX uses a more stringent termination criterion for the IPM when crossover is requested. In addition to (4) it is required that

$$\max_j \left\| \delta x_j A_j \right\|_\infty \leq 10^{-8}(1 + \|(\mathbf{b}, \mathbf{l}_{\mathscr{L}}, \mathbf{u}_{\mathscr{U}})\|_\infty), \tag{16a}$$

$$\max_j |\delta z_j| \leq 10^{-8}(1 + \|\mathbf{c}\|_\infty), \tag{16b}$$

where δx_j and δz_j are the perturbations to drop the iterate to complementarity. (For each j either δx_j or δz_j is zero.) On 60% of the models in our test set the stricter condition forced at least one extra interior point iteration, and except for some badly scaled models no more than 6 iterations were needed. Hence the additional computation time for the IPM is moderate, in particular because the final iterations are typically fast with basis preconditioning. Using the stricter termination criterion for the IPM, the basis after the push phases was optimal in 150 out of 165 cases.

A conventional IPM implementation would not be able to achieve (16) reliably because the linear systems would become too ill conditioned in the final iterations to be solved to sufficient accuracy. IPX dynamically eliminates variables from the optimization process when the primal is close to its bound or the dual is close to zero. This technique requires a basis to be implemented correctly and allows the IPM to be run to (arbitrary) high accuracy. Details can be found in [24].

6 Results

IPX is written in C++ and comprises about 10,000 lines of code for the core algorithm. The factorization of basis matrices and its update is separated from the interior point code and can be provided by an external package. By default the authors' BASICLU package [26] is used, which implements a Markowitz LU factorization and a variant of the Forrest-Tomlin update. Both packages are available from https://www.maths.ed.ac.uk/ERGO/software.html.

In this section the combination of IPX and BASICLU is tested on real-world LP models in terms of robustness and performance. Two benchmarks are made, one on a diverse set of LP models and one that considers specific problem classes. For comparison the interior point solver from the commercial software Gurobi [10] version 7.0 is used, which represents a state-of-the-art implementation based on Cholesky factorization. The computations were run on a desktop computer with an Intel i5-6500 processor with 4 cores and 8 GB physical memory. While it is clear that the Cholesky factorization would have benefitted from hardware with higher floating point capability more than the iterative solver, this setup was chosen because it is often used by practitioners.

6.1 Diverse test set

For the diverse test set all LP models from the sources listed in Appendix A were collected. For the mixed-integer problems the canonical LP relaxation was built, and a presolved version was generated for each model by the Gurobi LP presolve. Instances for which the resulting $\bar{m} \times \bar{n}$ constraint matrix was such that $\min\{\bar{m}, \bar{n}\} \leq 1000$ were removed, because for these models the normal equations (possibly after dualization) can be solved efficiently by dense linear algebra routines.

Both solvers were then applied with default parameters and a time limit of 36,000 s. IPX was run on the presolved models, whereas Gurobi was given the original models and its presolve time was subtracted from its total runtime. Infeasible and unbounded

Table 3 Dimensions of 170 test problems after presolve

$\min\{\bar{m}, \bar{n}\}$	Instances
1036–2499	16
2500–4999	17
5000–9999	27
10,000–24,999	41
25,000–49,999	31
50,000–99,999	11
100,000–249,999	19
250,000–499,999	4
500,000–999,999	3
1,000,000–1,439,571	1

models, as well as models for which the Cholesky factorization required more than the 8 GB physical memory were removed from the set. Models were also removed if they were solved by both methods within 1 s. The test set was finally cleaned by keeping only 1 or 2 instances of the same model; for example, from the 12 `pds` instances only `pds-60` and `pds-100` were kept. The resulting set contained 170 LP models that are listed with their solution times in Appendix A. An overview of their dimensions is given in Table 3. While there is a lack of truly large instances, the set represents a wide range of medium to large-scale models.

IPX does not provide a simplex implementation for cleaning up the basic solution after the crossover push phases. For the study the final basis was used as starting basis for the Gurobi primal or dual simplex, depending on which infeasibility was smaller. In 150 cases Gurobi decided the initial solution to be optimal within its default tolerances. In 15 cases a simplex run was necessary and the time was added to the total IPX runtime.

All models in the test set were solved by either IPX or Gurobi to basic solution. The Gurobi crossover reached time limit on 1 instance (`nug30`), whereas the IPX interior point method failed on 5 instances:

- On `stormg2_1000`, `ns2122603` and `ns1688926` the IPM stopped after making no progress over a number of iterations. The issue seems to be solvable by a refined IPM implementation (for example using centrality correctors or a more conservative choice of step sizes). The latter two models are questionable numerically, however, due to a wide range of entries in the problem data.
- On `cont1_l` and `cont11_l` the initial LU factorization ran out of memory. It turned out that the large fill-in was caused by the default pivot tolerance of 0.0625 being too small. For the related but smaller instances `cont1` and `cont11` IPX detected the initial LU factorization to be unstable and tightened the pivot tolerance to 0.3. In the repeated factorization the fill-in decreased by about a factor 4. After setting the initial LU pivot tolerance to 0.3, `cont1_l` was solved to an optimal basic solution in 3155 s (Gurobi required 1951 s); `cont11_l` reached time limit after 3 interior point iterations with basis preconditioning.

Table 4 Runtime comparison on 164 LP models that were solved by both methods

Subset	Instances	IPX/Gurobi	IPX faster	Gurobi faster
>1s	164	3.69	22	142
>10s	89	3.95	16	73
>100s	41	6.31	6	35

Table 4 compares the runtimes on the 164 models that were solved by both codes with default parameters. The column "IPX/Gurobi" shows the geometric mean of the runtime ratios, a value >1.0 meaning that IPX was by that factor slower. The subset ">10"s consists of the models for which at least one solver required more than 10 s. The last subset should be considered with care because 41 instances are insufficient to draw a conclusion.

IPX solved the vast majority of test problems and its average runtime on the medium-size instances was in the same order of magnitude as that of Gurobi. Hence the new approach proved to be a general-purpose LP solver. For large instances, say $m \geq 100{,}000$, the irregular memory access of the iterative solver and basis update scheme became decisive. Here the Cholesky factorization often performed better as long as the required number of floating point operations was not too high.

The breakdown of the total IPX runtime (without simplex clean-up) into different parts of the algorithm is illustrated in Fig. 2 for the 87 models that IPX solved successfully but took longer than 10 s. Computing the starting basis was inexpensive except for `cont1`, where it accounted for one third of the total time. Here the issue was the large fill-in in the first LU factorization before tightening the pivot tolerance. On average 15% of the time for running the linear solver was spent in the initial IPM iterations (not shown separately). Taking geometric means, preparing the preconditioner and running the iterative solver accounted for 22% and 45% of the total time, respectively. Crossover took 2% on average, but dominated the IPM runtime on 7 models.

6.2 Specific problem classes

The results from the previous subsection proved the robustness and general applicability of the new method, but indicated that on average the performance of IPX is inferior to that of a conventional, Cholesky-based IPM. However, there existed relevant models in the test set on which IPX was considerably faster than the Gurobi IPM, either because of the linear system solves in the IPM or because of the crossover. Two such LP models of which instances of different dimensions were available should be discussed.

The first example is a planning model from the petroleum industry that was formulated in [2]. The dimensions of the presolved models for 60, 120, 180, 240 and 300 planning days are given in Table 5. The models are hypersparse and `srd300` can be considered large-scale by today's standards.

The second example are LP relaxations of quadratic assignment problems, which are commonly used for modelling facility location problems. The `nug` models in

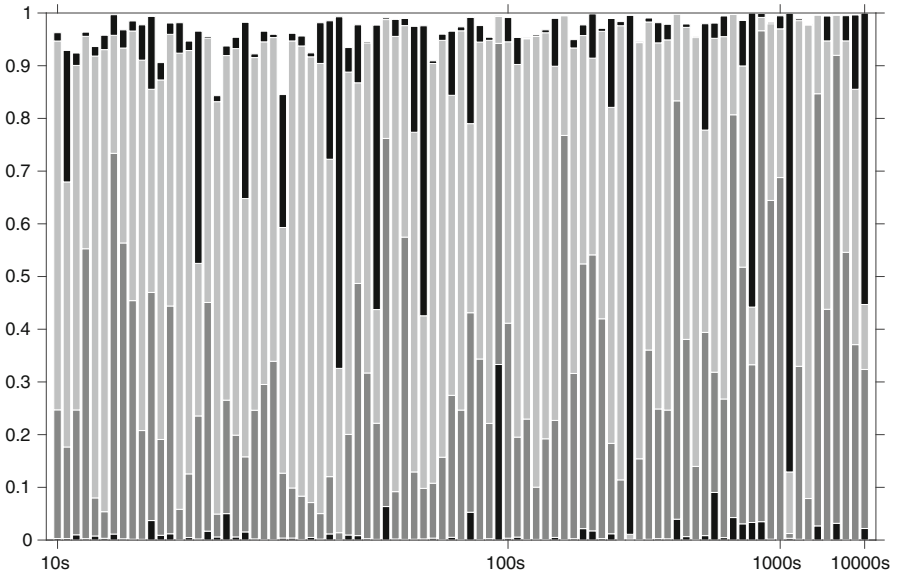

Fig. 2 Fraction of total runtime spent for computing the crash basis (black bar at the bottom), for updating the basis during the interior point solve (dark grey), for running the linear solver (light grey) and for crossover (black bar at the top). The white area corresponds to the runtime spent in the remaining parts of the algorithm. The 87 LP models are ordered on the x-axis by total runtime

Table 5 Dimensions of presolved LP models

Name	$\bar{m}$	$\bar{n}$	$\text{nnz}(\bar{A})$
srd060	93,200	178,510	2,796,210
srd120	186,440	357,070	9,804,510
srd180	279,680	535,630	21,024,810
srd240	372,920	714,190	36,457,110
srd300	466,160	892,750	56,101,410
nug12	2794	8771	33,443
nug15	5698	22,230	85,425
nug20	14,098	72,546	281,906
nug30	52,260	379,350	1,567,800

Table 5 originated from the binary LP formulations given in [22] of the facility location problems from [18] with 12, 15, 20 and 30 facilities. Although their dimensions are moderate, the models are known to be challenging for both Cholesky factorization and simplex.

In addition to the two interior point solvers, the Gurobi dual simplex was included in the comparison. The computations were performed as in the previous subsection, using default solver parameters and a time limit of 36,000 s. IPX was given the presolved models, whereas Gurobi was given the original models and the presolve time was subtracted afterwards.

Table 6 Computation times in seconds

Name	IPX			Gurobi IPM			Simplex
	Total	IPM	Crossover	Total	IPM	Crossover	
srd060	74.2	73.5	0.7	59.8	57.5	2.3	27.7
srd120	329.5	327.4	2.1	496.5	484.9	11.6	1048.3
srd180	891.4	886.8	4.6	1459.9	1435.8	24.1	3199.7
srd240	1883.3	1877.2	6.1	3615.5	3577.2	38.3	8483.8
srd300	3477.3	3467.1	10.2	(t)	(t)		14,433.2
nug12	2.4	2.3	0.1	0.6	0.5	0.2	16.4
nug15	15.1	13.3	1.8	4.0	2.8	1.2	238.5
nug20	192.3	176.1	16.1	435.5	37.5	398.0	3515.0
nug30	11,473.2	5125.2	6348.1	t	1400.5	t	t

"t" means 36,000 s time limit reached

The runtimes of the three solvers are given in Table 6. IPX terminated successfully on all problems and the basic solution was optimal except for nug30, where it was cleaned up by the primal simplex in 1 iteration. Both the Gurobi crossover and dual simplex reached time limit on nug30. The Gurobi IPM also reached time limit on srd300. The latter seemed to be caused by the use of swap memory which deteriorated the performance of the Cholesky factorization. We therefore ignore this result in the following discussion. For the other problems the size of the Cholesky factors was well within the 8 GB physical memory (see Table 7 below).

On the srd instances, except for the smallest one, IPX was between 3.2 and 4.5 times faster than the dual simplex and between 1.5 and 1.9 times faster than the Cholesky-based IPM. On this model the crossover times were almost negligible and the performance difference between the two interior point codes came from the cost of the linear system solves. On the nug models the Cholesky-based IPM without crossover was about 4 times faster than IPX without crossover. However, for the largest two instances, the basis at the end of the push phase of the Gurobi crossover was largely primal and dual infeasible. On nug20 the simplex clean-up took 49,068 iterations and 382 s, and on nug30 it reached time limit after 191,148 iterations and 18,693 s. On these problems the important feature of IPX is to compute an interior solution of high accuracy. When we used the standard 8-digit objective criterion (4) to terminate the IPM, then for nug30 the vertex solution defined by the basis at the end of the push phases violated the bound constraints by 10^{-3} and the dual sign condition by 10^{-4}. Neither the primal nor dual simplex finished within 36,000 s. By imposing the more stringent termination criterion (16), the IPM performed 4 additional iterations and the primal infeasibility of the final vertex solution decreased to 10^{-10}. The primal simplex then solved the model to optimality in 1 iteration.

In Table 7 the number of iterations performed by the Gurobi dual simplex are compared to the number of basis changes in IPX after construction of the starting basis. The latter include the basis updates during the IPM, the basis updates in the push phases and the iterations of the simplex clean-up (if any). On the srd models

Table 7 Factor size of the Gurobi Cholesky factorization, iteration count of the Gurobi dual simplex divided by $\bar{m}$, and number of basis updates in IPX including crossover divided by $\bar{m}$

Name	Cholesky	Simplex	IPX
srd060	400 MB	2.13	0.75
srd120	1.5 GB	2.21	0.75
srd180	3.0 GB	2.39	0.74
srd240	5.0 GB	2.57	0.79
srd300	8.0 GB	2.54	0.79
nug12	16 MB	12.54	0.71
nug15	60 MB	29.63	1.11
nug20	300 MB	39.99	1.00
nug30	4.0 GB	> 3.29	0.59

the dual simplex required roughly $2\bar{m}$ iterations, which is a common target that one would hope for. The method implemented in IPX is still clearly superior, both in terms of number of basis updates and computation time. On the nug models the number of simplex iterations grew quickly with the problem dimension. But this effect did not occur in IPX. Here the number of basis updates was at most $1.11\bar{m}$ and that ratio even decreased toward the larger instances.

7 Conclusions

The results show that a robust implementation of an interior point solver based on iterative linear algebra is possible. To our knowledge, IPX is the first code of that kind which is capable of solving a diverse set of medium to large-scale LP models reliably and accurately, and whose average runtime is in the same order of magnitude as that of a state-of-the-art direct method. The key idea of the approach presented in this paper is the maximum volume criterion for choosing a basis matrix for preconditioning. We proposed a heuristical method for maintaining a "good" basis at reasonable cost and showed empirically that the heuristically found basis is as effective as preconditioner as a maximum volume basis.

Maintaining a basis matrix in the IPM enables further options to improve numerical stability. We described the treatment of free variables, otherwise an issue in IPM implementation, and mentioned the ability to remove close-to-converged variables from the optimization process. The latter option allows running the IPM to high accuracy, which is required for a clean crossover. This technique can also be used to supplement direct linear algebra, by performing some extra IPM iterations with basis preconditioning prior to crossover.

Although our current implementation is slower than a Cholesky-based IPM on average, there exist relevant models on which it outperforms both a conventional IPM and a dual simplex. It can therefore be used as an alternative when appropriate. Furthermore, although we have spent about one year implementation effort into IPX, it is clear that the code is not as highly developed as commercial solvers are, and that there is certainly room for improvement.

Appendix A: Test set and solution times

The LP models have been obtained from the following sources:

(1) J. Castro: http://www-eio.upc.es/~jcastro/

   (a) huge CTA instances
   (b) integrated refinery problems
   (c) L1.zip
   (d) Linf.zip

(2) J. A. J. Hall: http://www.maths.ed.ac.uk/hall/PublicLP/
(3) Kennington collection: http://www.netlib.org/lp/data/kennington/index.html
(4) C. Mészáros: http://old.sztaki.hu/~meszaros/public_ftp/lptestset/

   (a) misc
   (b) New
   (c) problematic
   (d) stochlp

(5) MIPLIB 2010: http://miplib.zib.de/download/miplib2010-complete.tgz
(6) H. Mittelmann: http://plato.asu.edu/ftp/lptestset/

   (a) fome
   (b) misc
   (c) nug
   (d) pds
   (e) rail

(7) Netlib collection: http://www.netlib.org/lp/data/index.html

For mixed-integer problems, the canonical LP relaxation was used.

In the following table, the columns headed "$\bar{m}$", "$\bar{n}$" and "nnz" provide the number of rows, columns and nonzero entries of the constraint matrix $\bar{A}$ in the problem formulation (2a), after the LP model was preprocessed by the Gurobi LP presolve. The column block headed "IPX" provides the runtime of the IPX interior point solver ("IPM"), of the IPX crossover push phases ("push") and of the clean-up using the Gurobi simplex ("simplex"); the latter field is empty if no clean-up was needed. The column headed "total" gives the sum of the three runtimes. The column block headed "Gurobi" provides the runtimes of the Gurobi interior point solver ("IPM") and crossover ("crossover") as well as their sum ("total"). The presolve time has been excluded from these figures. Computation times are in seconds and were obtained on an Intel i5-6500 CPU (4 cores, 3.2 GHz, 6 MB L3 cache) and 8 GB of physical memory. Logfiles from IPX and Gurobi, from which the runtimes were extracted, are available at https://github.com/ERGO-Code/ipx/tree/master/benchmark.

src	Name	$\bar{m}$	$\bar{n}$	nnz	IPX				Gurobi		
					Total	IPM	Push	Cleanup	Total	IPM	Crossover
1a	L1_sixm250obs	154,168	308,284	639,901	180.1	176.5	3.6		43.5	42.1	1.4
1a	L1_sixm500obs	283,055	565,680	1,196,860	553.8	506.0	47.8		471.8	468.7	3.1
1b	srd120	186,440	357,070	9,804,510	329.5	327.4	2.1		496.5	484.9	11.6
1b	srd240	372,920	714,190	36,457,110	1883.3	1877.2	6.1		3615.5	3577.2	38.3
1c	L1_bts4	25,611	69,900	181,413	5.7	5.6	0.1		1.4	1.3	0.2
1c	L1_five20b	28,342	62,986	187,981	47.4	47.2	0.1		6.5	3.7	2.8
1d	Linf_bts4	61,465	70,128	287,954	11.1	10.8r	0.3	0.1	49.3	48.7	0.6
1d	Linf_five20b	60,911	63,458	285,418	243.5	64.6r	16.3	162.6	25.1	10.8	14.3
2	dcp2	13,927	25,376	281,550	5.7	5.7r	0.0		0.7	0.6	0.2
3	cre-b	5213	31,720	107,039	2.3	2.3	0.0		0.4	0.4	0.0
3	ken-18	39,855	89,346	204,936	5.5	5.5	0.1		1.3	1.1	0.2
3	osa-60	10,209	224,125	584,253	5.7	5.6	0.1		1.1	1.0	0.1
4a	bas1lp	5409	4443	582,390	4.0	3.8	0.2		1.2	1.0	0.1
4a	baxter	18,867	10,742	78,412	2.5	2.4	0.1		1.9	1.9	0.1
4a	co9	6784	10,675	81,809	2.1	2.1	0.0		0.4	0.4	0.1
4a	dbic1	33,598	140,205	781,668	58.4	46.7	11.7		12.0	2.8	9.2
4a	dbir1	7150	24,862	994,828	17.3	16.3	1.0		1.6	1.4	0.2
4a	e18	24,598	14,211	131,991	12.6	12.3	0.2		63.2	63.1	0.2
4a	ex3sta1	12,602	12,675	54,377	3.5	3.4	0.0		0.4	0.2	0.2
4a	jendrec1	2109	3535	88,915	0.9	0.9	0.0		1.1	0.7r	0.4
4a	lpl1	32,427	82,733	252,587	21.9	21.4	0.5		4.1	1.8	2.3
4a	mod2	23,706	24,005	114,068	6.1	6.1	0.0		1.5	1.2	0.2
4a	model10	2961	10,373	94,372	1.6	1.5	0.0	0.0	0.3	0.2	0.1
4a	nemsemm1	2829	36,052	495,916	5.8	5.8	0.0		0.6	0.6	0.1
4a	nl	5290	7997	36,524	1.0	1.0	0.0		0.4	0.4	0.0
4a	nsct1	7696	11,297	589,578	7.7	7.5	0.2		3.5	3.4	0.1
4a	p010	8907	17,770	60,150	4.3	4.2	0.0		0.1	0.1	0.0
4a	rat7a	2152	6772	156,249	2.0	2.0	0.0		0.5	0.3	0.2
4a	route	20,779	23,923	184,576	2.4	2.3	0.1		0.4	0.4	0.0
4a	stat96v1	4624	187,892	562,110	21.7	21.2r	0.4	0.1	20.8	4.9r	15.8

src	Name	$\bar{m}$	$\bar{n}$	nnz	IPX				Gurobi		
					Total	IPM	Push	Cleanup	Total	IPM	Crossover
4a	stat96v3	26,274	1,090,091	3,250,147	1423.9	470.1r	14.0	939.9	8063.5	5.2r	8058.3
4a	ulevimin	4594	41,225	135,479	3.9	3.9	0.0		0.6	0.5	0.1
4a	world	23,640	25,839	114,431	6.7	6.7	0.0		1.7	1.3	0.4
4b	degme	185,501	659,415	8,127,528	2496.8	2495.9	0.9		166.4	79.8	86.6
4b	karted	46,501	133,114	1,770,336	499.6	499.4	0.2		144.9	27.2	117.7
4b	tp-6	142,752	1,014,301	11,537,419	2820.9	2820.0	0.8		66.0	42.0	24.0
4b	ts-palko	22,002	47,235	1,076,903	314.7	314.5	0.2		79.2	8.6	70.6
4c	gen4	1475	4173	104,236	12.3	11.8r	0.5		1.8	0.5	1.3
4c	l30	2698	15,360	51,093	1.8	1.6	0.1	0.1	4.0	0.8	3.2
4d	fxm3_16	32,946	57,391	315,606	7.2	7.2	0.1		1.0	0.8	0.2
4d	pltexpa4_6	13,693	24,221	70,946	3.0	3.0	0.1		1.3	1.2	0.1
4d	scfxm1-2r-256	25,922	45,663	158,006	8.1	8.1	0.1		1.6	1.0	0.5
4d	stormg2-125	47,536	129,869	358,498	28.3	28.1	0.2		2.8	2.5	0.4
4d	stormg2_1000	380,036	1,038,119	2,865,373		f			29.3	24.1	5.2
5	30_70_45_095_100	12,515	10,967	46,614	2.4	1.0	1.5		1.2	0.3	0.9
5	app1-2	52,261	26,265	194,705	4.5	4.4	0.1		1.0	0.9	0.2
5	atlanta-ip	19,835	17,484	182,879	10.2	10.0	0.2		4.8	4.5	0.3
5	bab3	22,478	393,457	3,097,799	120.1	119.8	0.3		6.2	5.9	0.3
5	bley_xl1	175,178	5724	867,393	10.4	7.8	2.6		13.5	13.2	0.3
5	buildingenergy	225,031	128,695	683,844	262.5	260.5	2.0		6.1	5.7	0.5
5	circ10-3	42,620	2700	307,320	1.2	1.0	0.3		0.6	0.4	0.2
5	core4872-1529	4607	24,098	184,762	4.2	4.1	0.1		1.6	1.3	0.3
5	dano3mip	3150	13,837	79,530	1.7	1.7	0.0		1.1	0.9	0.2
5	datt256	9863	196,147	1,124,622	1218.3	158.1	1060.2		718.3	1.0	717.3
5	dc1l	1650	35,496	424,338	5.9	5.9	0.1		1.1	1.0	0.1
5	dolom1	1802	10,825	176,976	2.4	2.4	0.0		0.6	0.5	0.0
5	ds-big	1042	173,029	4,573,582	196.9	195.4	1.6		6.0	5.0	1.0
5	ex10	62,932	15,896	1,031,940	38.7	12.9	25.8		121.8	114.4	7.4
5	f2000	10,495	3995	29,490	3.6	2.3	1.3		2.8	0.9	1.9
5	germanrr	5524	10,650	159,548	2.0	1.9	0.0	0.0	0.3	0.3	0.0

src	Name	$\tilde{m}$	$\tilde{n}$	nnz	IPX				Gurobi		
					Total	IPM	Push	Cleanup	Total	IPM	Crossover
5	gmut-75-50	2499	35,915	569,806	6.0	5.9	0.0	0.0	0.8	0.7	0.1
5	in	1,495,549	1,439,571	6,696,217	5381.1	4622.9	758.2		101.2	82.7	18.4
5	ivu06-big	1177	2,197,774	22,556,378	2067.8	2064.9	2.9		34.3	30.7	3.6
5	ivu52	2116	157,543	2,178,871	69.2	68.4	0.8		3.5	3.2	0.3
5	map06	37,504	18,973	81,653	11.3	11.2	0.1		3.5	3.2	0.3
5	mining	661,094	348,958	2,754,430	772.0	770.0	2.0		46.6	36.8	9.8
5	momentum3	56,421	13,334	566,199	25.7	25.6	0.2		22.1	18.0	4.0
5	msc98-ip	15,293	12,823	81,666	4.1	3.6	0.3	0.2	3.0	2.6	0.4
5	mspp16	524,814	4081	27,555,511	65.3	65.0	0.3		57.4	52.7	4.7
5	mzzv11	9316	9902	133,424	2.3	2.1	0.2		1.6	1.5r	0.1
5	n15-3	28,860	153,140	575,246	19.0	18.6	0.3		2.4	2.1	0.3
5	n3seq24	5950	119,856	2,404,844	27.0	26.5	0.5		4.1	3.7	0.3
5	nb10tb	94,742	49,744	802,185	92.8	78.2	0.6	14.0	53.6	41.9	11.7
5	neos-1140050	3795	39,075	806,835	6.8	6.8	0.0		3.9	3.3	0.6
5	neos-1429212	29,416	64,404	1,222,493	34.1	31.4	2.6		3.3	1.9	1.4
5	neos-1605075	3464	4085	91,314	1.3	0.9	0.3		1.0	0.9	0.2
5	neos-476283	4312	6528	3,924,503	23.8	23.3	0.5		20.0	19.8	0.3
5	neos-506428	129,925	42,981	343,466	7.5	4.4	3.1		1.3	1.0	0.3
5	neos-520729	17,391	64,464	175,131	12.1	11.8	0.3		0.4	0.4	0.1
5	neos-631710	169,576	167,056	834,166	220.2	3.4	216.8		8.7	1.2	7.5
5	neos-738098	25,736	8981	100,842	2.6	1.5	1.1		0.6	0.5	0.1
5	neos-799711	12,660	11,615	40,152	1.4	1.3	0.1		0.5	0.4	0.1
5	neos-824661	18,804	29,920	122,230	2.0	1.7	0.3		0.3	0.2	0.1
5	neos-826694	6730	15,210	52,020	1.2	0.8	0.3		0.2	0.1	0.1
5	neos-933638	9642	8887	54,484	1.9	1.0	0.9		0.6	0.4	0.3
5	neos-941313	13,099	116,670	386,400	8.4	6.9	1.6		1.1	0.4	0.7
5	neos-948126	7208	7777	37,891	1.6	1.1	0.5		0.4	0.2	0.2
5	neos-957389	5114	6034	355,369	4.4	4.3	0.1		0.6	0.5	0.1
5	neos-984165	6921	7315	36,573	1.7	1.2	0.5		0.4	0.2	0.2
5	neos6	1036	8563	251,723	1.1	1.1	0.0		0.5	0.4	0.0
5	neos808444	17,729	19,330	119,357	2.8	1.2	1.6		0.6	0.4	0.3

src	Name	$\bar{m}$	$\bar{n}$	nnz	IPX				Gurobi		
					Total	IPM	Push	Cleanup	Total	IPM	Crossover
5	net12	13,975	14,115	80,108	2.8	2.8	0.0		4.4	4.4	0.0
5	netdiversion	99,581	128,968	495,878	110.3	105.2	5.1		14.4	4.2	10.3
5	npmv07	60,835	162,180	378,798	23.1	22.9	0.2		3.7	3.5	0.2
5	ns1111636	13,453	76,462	283,284	5.3	5.1	0.2		0.5	0.4	0.1
5	ns1116954	131,865	11,928	409,850	7.0	2.1	4.9		12.2	11.7	0.4
5	ns1631475	24,074	22,485	100,023	4.6	4.4	0.3		1.0	0.5	0.5
5	ns1644855	30,597	30,199	2,100,495	49.5	47.9	1.6		131.6	131.2	0.5
5	ns1663818	167,047	123,027	20,177,986	1105.0	1077.8r	27.3		208.3	197.3r	10.9
5	ns1685374	43,195	9174	206,101	9.9	9.9	0.0		11.9	1.6	10.4
5	ns1696083	10,513	7748	371,757	4.9	4.9	0.1		1.7	1.6	0.1
5	ns1758913	615,190	17,824	1,265,492	26.2	19.6	6.6		21.0	3.9	17.1
5	ns1853823	223,144	213,176	1,346,924	492.6	473.3	19.3		32.7	20.8r	11.9
5	ns1854840	143,236	135,754	844,834	136.5	124.1	12.4		6.2	4.7	1.6
5	ns1904248	146,398	38,262	371,070	6.6	1.7	4.9		0.8	0.7	0.2
5	ns1905797	51,876	18,188	239,156	1.6	1.5	0.1		11.4	11.3	0.1
5	ns2017839	49,884	53,364	298,482	20.2	20.1	0.1	0.0	7.3	3.2r	4.1
5	ns2118727	160,408	164,350	636,614	42.0	41.9	0.1		13.8	13.5	0.3
5	ns2122603	19,036	16,556	64,357		f			4.4	1.6	2.9
5	ns2124243	32,277	48,893	192,513	4.8	4.4	0.4		0.6	0.3	0.3
5	ns2137859	99,385	99,462	593,332	11.6	11.4	0.2		2.7	2.1	0.5
5	ns894244	9479	16,399	68,362	3.1	2.6	0.5		0.8	0.6	0.2
5	ns930473	22,846	33,136	139,974	7.2	6.8	0.4		1.1	0.8	0.4
5	nsr8k	6283	38,264	370,206	16.8	16.4	0.4		4.7	3.9	0.7
5	ofi	105,107	222,217	848,037	135.5	134.7	0.8	0.1	29.0	28.5	0.5
5	opm2-z11-s8	205,936	7653	473,063	19.4	10.9	8.6		22.4	15.0	7.4
5	opm2-z12-s7	297,892	10,336	678,441	44.4	20.4	23.9		37.8	23.2	14.6
5	pb-simp-nonunif	123,532	11,798	298,082	1.3	1.0	0.3		1.3	0.7	0.5
5	rail02	54,524	192,618	599,436	71.0	62.3	8.6		45.2	42.0	3.2
5	rail03	129,647	567,095	1,349,518	399.2	318.7	80.5		55.4	48.7	6.8

src	Name	$\bar{m}$	$\bar{n}$	nnz	IPX				Gurobi		
					Total	IPM	Push	Cleanup	Total	IPM	Crossover
5	ramos3	2187	2187	32,805	4.6	1.0	3.6		4.4	0.3	4.1
5	reblock420	62,800	4200	138,670	1.9	1.2	0.8		0.8	0.6	0.3
5	rmatr100-p5	8685	8784	26,152	3.9	3.8	0.0		9.1	9.1	0.0
5	rmatr200-p5	37,617	37,816	113,048	56.5	55.7	0.8		1.3	1.2	0.1
5	rmine14	268,474	32,144	659,980	83.0	80.4	2.6		48.6	20.8	27.8
5	rmine21	1,441,506	162,402	3,514,014	2707.8	2580.2	127.7		1620.0	694.8	925.2
5	rocII-9-11	42,412	20,557	503,343	6.6	6.3	0.3		1.1	0.7	0.3
5	satellites3-40-fs	28,171	54,578	199,798	22.0	14.7	7.4		6.2	3.1	3.1
5	satellites3-40	37,422	54,578	588,630	34.4	25.4	9.0		61.2	56.6	4.6
5	sct1	9171	14,514	85,676	4.2	4.1	0.1		1.3	1.2	0.1
5	shs1023	126,657	432,068	1,013,029	173.5	170.8	2.7		14.4	12.8	1.6
5	siena1	2211	13,482	254,310	3.9	3.8	0.1		1.1	1.0	0.1
5	sing161	150,806	464,036	1,459,828	281.9	273.6	8.3		13.6	12.1	1.5
5	sing359	153,253	428,748	1,408,257	268.7	258.4	10.3		11.9	10.1	1.8
5	sp97ar	1692	14,100	281,317	1.4	1.4	0.0		0.2	0.2	0.0
5	splan1	521,819	1,253,087	5,088,694	5242.2	4989.8	250.4	1.9	1775.7	1587.6	188.1
5	stockholm	36,517	16,622	111,003	5.6	5.5	0.1		0.7	0.6	0.1
5	stp3d	97,936	137,646	500,753	196.5	163.4	33.1		11.2	9.4	1.8
5	tanglegram1	68,210	34,501	204,158	7.3	5.8	1.5		7.7	0.3	7.4
5	triptim3	14,121	22,982	494,503	14.2	13.3	0.9		5.3	4.8	0.5
5	uc-case3	34,288	27,194	230,068	7.2	7.0	0.2		1.1	0.9	0.2
5	unitcal_7	43,301	25,767	109,859	12.5	12.1	0.4		1.1	0.9	0.2
5	van	22,016	7360	481,408	2.7	2.7	0.0		7.5	7.3	0.1
5	vpphard	39,140	43,395	332,904	8.2	7.2	1.0		3.7	3.4	0.3
5	vpphard2	136,399	139,234	561,958	26.2	25.8	0.4		36.6	36.3	0.3
5	wnq-n100-mw99-14	656,900	10,000	1,333,400	20.3	20.0	0.2		39.4	39.0	0.3
6a	fome13	34,600	76,249	246,895	40.5	36.0	4.5		5.8	4.6	1.2
6a	fome21	24,173	162,336	356,706	15.5	15.0	0.5		3.5	3.1	0.4
6b	cont1	120,395	40,398	359,593	88.5	88.4	0.1		18.8	2.0	16.8

src	Name	$\tilde{m}$	$\tilde{n}$	nnz	IPX				Gurobi		
					Total	IPM	Push	Cleanup	Total	IPM	Crossover
6b	cont11	120,395	80,396	359,593	668.6	641.7	4.8	22.1	250.0	1.9	248.1
6b	cont11_l	1,468,599	981,396	4,403,001		f			3256.3	32.1	3224.2
6b	cont1_l	1,918,399	641,598	5,752,001		f			1951.3	50.3	1900.9
6b	neos	419,478	41,140	911,651	118.8	118.6	0.2		8.1	7.8	0.3
6b	neos1	131,581	1892	468,009	7.9	7.8	0.1		1.4	1.3	0.1
6b	neos3	512,209	6624	1,542,816	60.9	27.4	33.5		38.6	11.4	27.2
6b	ns1687037	36,080	30,955	1,377,655	145.8	145.8	0.1		24.5	15.4	9.1
6b	ns1688926	24,576	16,489	901,120		f			28.9	22.5r	6.4
6b	sgpf5y6	19,499	39,020	109,247	3.4	3.3	0.1		1.7	1.4	0.3
6b	watson_1	107,522	216,565	658,933	85.7	85.2	0.5		4.5	3.3	1.3
6b	watson_2	185,474	378,986	1,040,238	193.8	192.7	1.1		6.5	3.5	3.0
6c	nug08-3rd	18,270	20,448	128,547	573.8	253.7	320.1		91.2	18.8	72.4
6c	nug20	14,098	72,546	281,906	192.3	176.1	16.1		435.5	37.5	398.0
6c	nug30	52,260	379,350	1,567,800	11,473.2	5125.2	6337.0	11.1		1400.5	t
6d	pds-100	94,994	433,867	933,313	110.6	104.9	5.7		30.9	28.1	2.7
6d	pds-60	54,289	285,112	617,497	39.5	37.7	1.8		16.5	15.4	1.1
6e	rail2586	2463	909,940	7,901,059	259.0	258.6	0.4		15.3	12.7	2.7
6e	rail4284	4176	1,090,526	11,174,639	370.7	369.9	0.8		32.9	29.7	3.2
7	dfl001	4325	9511	30,833	2.2	2.0	0.1		0.6	0.5	0.1
7	pilot87	1815	4420	70,035	1.6	1.6	0.0		0.4	0.3	0.2
7	qap15	5698	22,218	85,413	15.4	13.3	2.1		4.3	2.8	1.5

f failed, t 36,000 s time limit reached, r IPM solution reported not optimal

References

1. Al-Jeiroudi, G., Gondzio, J., Hall, J.A.J.: Preconditioning indefinite systems in interior point methods for large scale linear optimisation. Optim. Methods Softw. **23**(3), 345–363 (2008)
2. Alabi, A., Castro, J.: Dantzig–Wolfe and block coordinate-descent decomposition in large-scale integrated refinery-planning. Comput. Oper. Res. **36**(8), 2472–2483 (2009)
3. Arioli, M., Duff, I.S.: Preconditioning linear least-squares problems by identifying a basis matrix. SIAM J. Sci. Comput. **37**(5), S544–S561 (2015)
4. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice—closing the gap. In: System Modelling and Optimization: Methods, Theory and Applications, pp. 19–50. Kluwer Academic Publishers (1999)
5. Bixby, R.E., Saltzman, M.J.: Recovering an optimal LP basis from an interior point solution. Oper. Res. Lett. **15**(4), 169–178 (1994)
6. Duff, I.S.: On algorithms for obtaining a maximum transversal. ACM Trans. Math. Softw. **7**(3), 315–330 (1981)
7. Fong, D.C.L., Saunders, M.A.: CG versus MINRES: an empirical comparison. SQU J. Sci. **17**(1), 44–62 (2012)
8. Gondzio, J.: Multiple centrality corrections in a primal-dual method for linear programming. Comput. Optim. Appl. **6**(2), 137–156 (1996)
9. Goreinov, S.A., Oseledets, I.V., Savostyanov, D.V., Tyrtyshnikov, E.E., Zamarashkin, N.L.: How to find a good submatrix. In: Matrix Methods: Theory. Algorithms and Applications, pp. 247–256. World Sci. Publ, Hackensack, NJ (2010)
10. Gurobi Optimization. http://www.gurobi.com. Accessed 24 Jan 2017
11. Hall, J.A.J., McKinnon, K.I.M.: Hyper-sparsity in the revised simplex method and how to exploit it. Comput. Optim. Appl. **32**(3), 259–283 (2005)
12. Harris, P.M.J.: Pivot selection methods of the Devex LP code. Math. Program. **5**, 1–28 (1973)
13. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving inear systems. J. Res. Natl. Bureau Stand. **49**(6), 409–436 (1952)
14. Higham, N.J., Relton, S.D.: Estimating the largest elements of a matrix. SIAM J. Sci. Comput. **38**(5), C584–C601 (2016)
15. Knuth, D.E.: Semioptimal bases for linear dependencies. Linear Multilinear Algebra **17**(1), 1–4 (1985)
16. Megiddo, N.: On finding primal- and dual-optimal bases. ORSA J. Comput. **3**(1), 63–65 (1991)
17. Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM J. Optim. **2**(4), 575–601 (1992)
18. Nugent, C.E., Vollman, T.E., Ruml, J.: An experimental comparison of techniques for the assignment of facilities to locations. Oper. Res. **16**, 150–173 (1968)
19. Oliveira, A.R.L., Sorensen, D.C.: A new class of preconditioners for large-scale linear systems from interior point methods for linear programming. Linear Algebra Appl. **394**, 1–24 (2005)
20. Paige, C.C., Saunders, M.A.: Solution of sparse indefinite systems of linear equations. SIAM J. Numer. Anal. **12**(4), 617–629 (1975)
21. Pan, C.T.: On the existence and computation of rank-revealing LU factorizations. Linear Algebra Appl. **316**, 199–222 (2000)
22. Resende, M.G.C., Ramakrishnan, K.G., Drezner, Z.: Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. Oper. Res. **43**(5), 781–791 (1995)
23. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2003)
24. Schork, L.: Basis Preconditioning in Interior Point Methods. Ph.D. thesis, University of Edinburgh (2018)
25. Schork, L., Gondzio, J.: Maintaining a basis matrix in the linear programming interior point method. Tech. Rep. ERGO-17-009, University of Edinburgh (2017)
26. Schork, L., Gondzio, J.: Permuting spiked matrices to triangular form and its application to the Forrest-Tomlin update. Tech. Rep. ERGO-17-002, University of Edinburgh (2017)

27. Wright, S.J.: Primal-Dual Interior-Point Methods. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (1997)