McMaster University

Advanced Optimization Laboratory

Title:

On Implementing Self-Regular Proximity Based Feasible IPMs

Author:

Xiaohang Zhu Jiming Peng Tamás Terlaky Guoqing Zhang

AdvOl-Report No. 2004/3

April 2004, Hamilton, Ontario, Canada

ON AN IMPLEMENTATION OF SELF-REGULAR PROXIMITY BASED FEASIBLE IPMS

XIAOHANG ZHU†, JIMING PENG†, TAMÁS TERLAKY†, AND GUOQING ZHANG‡

Abstract. Self-regular based interior point methods present a unified novel approach for solving linear optimization and conic optimization problems. So far it was not known if the new Self-Regular IPMs can lead to similar advances in computational practice as shown in the theoretical analysis. In this paper, we present our experiences in developing the software package McIPM that uses the dynamic version of SR IPMs based on the homogeneous self-dual embedding model. After a brief review of the underlying algorithm, various issues with respect to implementation are addressed. Numerical stability and sparsity of the normal equation system are explored as well. Extensive testing shows that the McIPM package is competitive with other leading academic packages and that the Self-Regular proximity based approach allows to improve the performance of interior point method software when solving difficult linear optimization problems, and offers avenues for further improvements.

Key words. Linear optimization, interior point methods, self-regular proximity, self-dual embedding.

1. Introduction. Linear Optimization (LO) involves minimizing a linear objective subjected to some constraints over its variables. It has a broad range of applications in the field of operation research and many other areas of science and engineering. In practice, LO problems are usually very large in terms of the numbers of variables and constraints. Therefore it is important to develop new tools that enhance our capacity to solve large-scale LO problems.

Interior-point methods provide a powerful tool for solving LO problems. The age of IPMs started in 1984 when Karmarkar announced his seminal work and claimed that his algorithm enjoys a polynomial $O(nL)$ iterations complexity with a total $O(n^{3.5}L)$ bit operations and performs much better than the Simplex method, at that time the standard choice for solving LO problems. Karmarkar's result [14] sparked a surge of research on interior-point methods. Over the past decades, IPMs for LO have gained extraordinary interest and the topic has been developed into a mature discipline. Nowadays, IPMs has become as an alternative to simplex based methods and implemented in both academic and commercial optimization packages for solving LO problems. This is particularly useful in applications where the size of the underlying LO problem is very large.

Most IPMs can be categorized into two classes: the large-update and small-update IPMs depending on the strategies used in the algorithm to update the parameter regarding the duality gap. It has been proved that the small-update IPMs has the best known worst-cast complexity result ($O\left(\sqrt{n}\log\frac{n}{\varepsilon}\right)$) while the large-update IPMs perform much better in practice with an $O\left(n\log\frac{n}{\varepsilon}\right)$ complexity.

Recently, a new framework for the design and analysis of primal-dual IPMs was introduced by Peng, Roos, and Terlaky [25]. The approach in [25] is based on the notion of Self-Regular proximity (SR-proximity) that can be extended to the case of LO over self-dual cones. One important feature of the new class of Self-Regular IPMs (SR-IPMs) is that the theoretical worst-case complexity of large-update SR-IPMs is $O\left(\sqrt{n}\log n\log\frac{n}{\varepsilon}\right)$, that is almost a factor of $\sqrt{n}$ better than the complexity of classical large-update IPMs, while small-update SR-IPMs preserve the same iteration bound as

†Advanced Optimization Laboratory, Dept. Computing & Software, McMaster University.
‡Dept. of Industrial & Manufacturing Systems Engineering, University of Windsor.

that of existing small-update IPMs. It should be mentioned that although SR-IPMs follows that same framework as classical IPMs, it was not clear if the theoretical advantage of SR-IPMs would lead to computational efficiency in practice.

The main goal of this work is to explore the advantages of using SR-IPMs in computational practice. For this purpose we develop a practical variant of the dynamic SR-IPM [26] that improve on the performance of classical IPMs, in particular when solving large-scale, difficult LO problems. In the dynamic algorithm the barrier order of self regular proximities is chosen adaptively. At each iteration first the classical search direction is tried. If the classical search direction does not provide sufficient improvement, the barrier degree of the SR proximity is dynamically increased and, after a successful step, the barrier degree is reset to the default value.

For credible, rigorous evaluation of the efficiency of SR-IPMs in computational practice a reliable, robust and easy to modify software package is needed. Therefore we decided to develop our own software package, the so-called McIPM package, that enabled us to experiment freely. This paper contains a thorough description of the algorithmic framework of our implementation and all important aspects and elements of a powerful IPM software package. We discuss several implementation issues, e.g., how to design an algorithm that combines the advantages of SR search directions and powerful heuristics, such as Mehrotra-type predictor-corrector strategies. Further, sparsity and numerical stability of the normal equation system, line-search and stopping criteria are discussed. The McIPM software is based on the embedding model [32] that allows us to conduct our experiments with feasible SR-IPMs.

The software package McIPM was developed in MATLAB and C with calls to LIPSOL's "preprocess" and "postprocess" subroutines. The sparse matrix package WSMP[1] [11] is used to solve large sparse linear systems of equations at each iteration. We have tested McIPM on all the Netlib LO testing sets that has become the standard benchmark for testing and comparing LO algorithms and softwares. Our computational experiences demonstrate that McIPM is competitive with state of the art academic packages. Extensive testing shows that SR-IPMs are not only of theoretical interest but also offer avenues to enhance the efficiency of IPM based software packages.

The paper is organized as follows. In Section 2, we introduce some notation and give a brief description of path-following IPMs. Some details of the self-dual model are presented too. In Section 3, we describe the general framework of SR-IPMs. More details about dynamic SR-predictor-corrector IPM, are presented in Section 4. Several major implementation issues are discussed in Section 5. Numerical results are presented in Section 6 followed by concluding remarks in Section 7.

2. Linear Optimization Problems. We consider the following LO problem:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t} & Ax = b, \\ & 0 \le x_i \le u_i, \quad i \in \mathcal{I}, \\ & 0 \le x_j, \qquad\ \ j \in \mathcal{J}, \end{array}$$

where $A \in \mathcal{R}^{m\times n}$, $c, x \in \mathcal{R}^n$, $b \in \mathcal{R}^m$, $u_i \in \mathcal{R}$, with $\mathcal{I}$ with $\mathcal{J}$ are index sets such that $\mathcal{I} \bigcup \mathcal{J} = \{1, 2, \ldots, n\}$ and $\mathcal{I} \bigcap \mathcal{J} = \emptyset$. Without loss of generality, the last two constraints can be written as

$$Fx + s = u,\ x \ge 0,\ s \ge 0, \quad F \in \mathcal{R}^{m_f \times n},$$

[1] http://www.cs.umn.edu/~agupta/wsmp.html

where $m_f = |\mathcal{I}|$ and $s \in \mathcal{R}^{m_f}$ is the slack vector, the rows of F are unit vectors, and $u = (u_1, u_2, \cdots, u_{m_f})^T$. Hence, the above LO problem can be written as:

$$\begin{array}{lll} \min & c^T x & \\ \text{s.t} & Ax & = b, \\ & Fx + s & = u, \\ & x, s \geq 0. & \end{array} \tag{P}$$

The dual of problem (P) can be written as:

$$\begin{array}{ll} \max & b^T y - u^T w \\ \text{s.t} & A^T y - F^T w + z = c, \\ & w, z \geq 0, \end{array} \tag{D}$$

where $y \in \mathcal{R}^m$, $w \in \mathcal{R}^{m_f}$ and $z \in \mathcal{R}^n$.

Optimal solutions of the primal and dual LO problems (P) and (D), if they exist, satisfy the following optimality conditions (or KKT conditions):

$$\begin{array}{rlll} Ax = & b, & x \geq 0, \\ Fx + s = & u, & s \geq 0, \\ A^T y + z - F^T w = & c, & z \geq 0, \\ Xz = & 0, & \\ Sw = & 0, & \end{array} \tag{2.1}$$

where X and S are diagonal matrices with diagonal elements x_j and s_j, respectively. The equations $Xz = 0$ and $Sw = 0$ represent the complementarity conditions. The quantity $x^T z + s^T w$ is called the duality gap [27].

2.1. Primal-dual IPMs. The central path for primal-dual path-following IPMs is defined as the solution set $\{(y(\mu), w(\mu), x(\mu), s(\mu), z(\mu)) \mid \mu > 0\}$ of the system

$$\begin{array}{rlll} Ax = & b, & x \geq 0, \\ Fx + s = & u, & s \geq 0, \\ A^T y + z - F^T w = & c, & z \geq 0, \\ Xz = & \mu e, & \\ Sw = & \mu e, & \end{array} \tag{2.2}$$

where $e = (1, 1, \cdots, 1)^T$ is a vector of one's in appropriate dimension.

Given a feasible interior point $x, z \in \mathcal{R}^n_{++}$, $s, w \in \mathcal{R}^{m_f}_{++}$, $y \in \mathcal{R}^m$, where $\mathcal{R}^n_+$ is the nonnegative and $\mathcal{R}^n_{++}$ is the positive orthant in $\mathcal{R}^n$, Newton's direction is obtained by solving the following system of linear equations

$$\begin{pmatrix} 0 & 0 & A & 0 & 0 \\ 0 & 0 & F & I & 0 \\ A^T & -F^T & 0 & 0 & I \\ 0 & 0 & Z & 0 & X \\ 0 & S & 0 & W & 0 \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta w \\ \Delta x \\ \Delta s \\ \Delta z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mu e - Xz \\ \mu e - Sw \end{pmatrix}. \tag{2.3}$$

System (2.3) is called the primal-dual Newton system. If the matrix A has full row rank, then the coefficient matrix of system (2.3) is nonsingular and therefore it has a

unique solution that defines the classical Newton direction that used in primal-dual path-following IPMs.

Let us define

$$v = \left(\sqrt{\frac{x_1 z_1}{\mu}}, \sqrt{\frac{x_2 z_2}{\mu}}, \cdots, \sqrt{\frac{x_n z_n}{\mu}}, \sqrt{\frac{s_1 w_1}{\mu}}, \sqrt{\frac{s_2 w_2}{\mu}}, \cdots, \sqrt{\frac{s_{m_f} w_{m_f}}{\mu}} \right)^T.$$

The classical primal-dual logarithmic-barrier function for LO is defined by

$$\Psi(v) = \sum_{i=1}^{n+m_f} (v_i^2 - 1 - 2\log v_i), \; v_i > 0,$$

This proximity function can be used to measures the distance from the present point to the central path. A general scheme of primal-dual path-following IPMs can be stated as follows [27].

Primal-Dual Path-Following IPMs

Input:
A proximity parameter $\delta > 0$;
an accuracy parameter $\varepsilon > 0$;
a fixed barrier update parameter θ, $0 < \theta < 1$;
a feasible interior point $(y^0, w^0, x^0, s^0, z^0)$ and $\mu^0 = 1$ such that $\Psi(v^0) \leq \delta$.
begin
$(y, w, x, s, z) := (y^0, w^0, x^0, s^0, z^0)$, $\mu := \mu^0$.
while $x^T z + s^T w \geq \varepsilon$ **do** (outer iteration)
begin
$\mu := (1 - \theta)\mu$.
while $\Psi(v) \geq \delta$ **do** (inner iteration)
begin
Solve (2.3) for $\Delta y, \Delta w, \Delta x, \Delta s, \Delta z$;
Compute a step size[2] α that sufficiently decreases $\Psi(v)$;
Update the iterate by:
$(y, w, x, s, z) := (y, w, x, s, z) + \alpha(\Delta y, \Delta w, \Delta x, \Delta s, \Delta z)$.
end
end
end

It has been show that the worst-case complexity of this algorithm is $O(n \log \frac{n}{\varepsilon})$ [27].

2.2. Self-Dual Embedding Model. Many Primal-dual IPMs need a strictly feasible initial solution to start with, and they work with LO problems for which a primal-dual optimal solution exists. However, there exist LO problems that does not possess an interior feasible solution. When we develop a solver for LO, we need to deal with two important issues: initialization and infeasibility detection. One elegant answer for both questions is the self-dual embedding model [32], which is the model used in our package.

[2]The step size has to be computed so that the next iterate is in the interior of the feasible region.

The self-dual embedding model first transfers the original problem (P) into a slightly larger LO problem, called the augmented problem that always has an optimal solution. The optimal solution of the augmented problem either proves that the original problem does not have an optimal solution or can easily be converted to an optimal solution of the original problem. The embedding problem can be formulated as follows.

Let $x^0, z^0 \in \mathcal{R}^n_{++},\ w^0, s^0 \in \mathcal{R}^{m_f}_{++},\ y^0 \in \mathcal{R}^m$, and $\tau^0, \nu^0, \kappa^0 \in \mathcal{R}_{++}$ be a given initial interior (possible infeasible) solutions. Define $r_{\rm P1} \in \mathcal{R}^m,\ r_{\rm P2} \in \mathcal{R}^{m_f},\ r_{\rm D} \in \mathcal{R}^n$ be the scaled errors at this initial interior solutions, and let $r_{\rm G} \in \mathcal{R},\ \beta \in \mathcal{R}$ be parameters defined as follows:

$$\begin{aligned}
r_{\rm P1} &= \frac{(Ax^0 - b\tau^0)}{\nu_0},\\
r_{\rm P2} &= \frac{(-Fx^0 + u\tau^0 - s^0)}{\nu^0},\\
r_{\rm D} &= \frac{F^T w^0 + c\tau^0 - A^T y^0 - z^0}{\nu^0},\\
r_{\rm G} &= \frac{-u^T w^0 + b^T y^0 - c^T x^0 - \kappa^0}{\nu^0},\\
\beta &= -r_{\rm P1}^T y^0 - r_{\rm P2}^T w^0 - r_{\rm D}^T x^0 - r_{\rm G}\tau^0,\\
&= \frac{(x^0)^T z^0 + (s^0)^T w^0 + \tau^0\kappa^0}{\nu^0} > 0.
\end{aligned}$$

The self-dual embedding model can be written as

$$\begin{array}{lrcrcrcrcrcrcrcl}
\min & & & & & & & & \beta\nu & & & & & & \\
\text{s.t.} & & & & & Ax & - & b\tau & - & r_{\rm P1}\nu & & & & = 0, \\
& & & - & Fx & + & u\tau & - & r_{\rm P2}\nu & -s & & & & = 0, \\
& -A^T y & + & F^T w & & & + & c\tau & - & r_{\rm D}\nu & & -z & & = 0, \\
& b^T y & - & u^T w & - & c^T x & & & - & r_{\rm G}\nu & & & -\kappa & = 0, \\
& r_{\rm P1}^T y & + & r_{\rm P2}^T w & + & r_{\rm D}^T x & + & r_{\rm G}\tau & & & & & & = -\beta, \\
& y, & \nu & \text{free}, & w \ge 0, & x \ge 0, & \tau \ge 0, & z \ge 0, & s \ge 0, & \kappa \ge 0. & & & & &
\end{array} \tag{SP}$$

The first four constraints in (SP), with $\tau = 1$ and $\nu = 0$, represent primal and dual feasibility (with $x \ge 0, s \ge 0, w \ge 0, z \ge 0$), and the reversed weak duality inequality, that is associated with primal and dual optimal solutions for (P) and (D). An artificial variable ν with appropriate coefficient vectors is added to achieve feasibility of (x^0, s^0) and $(y^0,\ w^0, z^0)$, and the fifth constraint is added to ensure self-duality. It is easy to see that problem (SP) is a skew-symmetric and self-dual LO problem [27, 32, 35]. We can easily verify the following lemma.

LEMMA 2.1. *The initial solution* $(y^0,\ w^0,\ x^0,\ \tau^0,\ \nu^0,\ s^0,\ z^0,\ \kappa^0)$ *is interior feasible for problem (SP). Moreover, if we choose* $y^0 = 0,\ \ w^0 = e,\ x^0 = e,\ \tau^0 = 1,\ \nu^0 = 1,\ s^0 = e,\ z^0 = e,\ \kappa^0 = 1$, *then this solution is a perfectly centered initial solution for problem (SP) with* $\mu = 1$.

Let $\mathcal{F}_{\mathcal{SP}}$ denotes the set of feasible solutions of problem (SP). We have

LEMMA 2.2. *If* $(y,\ w,\ x,\ \tau,\ \nu,\ s,\ z,\ \kappa) \in \mathcal{F}_{SP}$ *then*

$$x^T z + s^T w + \tau\kappa = \nu\beta. \tag{2.4}$$

Proof. Multiplying the first, second, third, fourth and fifth equality constraints by y^T, w^T, x^T, τ and ν, respectively, and summing them up, we have

$$x^T z + s^T w + \tau\kappa = \nu\beta.$$

The lemma is proved. □

THEOREM 2.3. *[27, 32, 35] The self-dual problem (SP) has the following properties:*

(i) *Problem (SP) has an optimal solution, and its optimal solution set is bounded.*

(ii) *The optimal value of (SP) is zero. If* $(y^*,\ w^*,\ x^*,\ \tau^*,\ \nu^*,\ s^*,\ z^*,\ \kappa^*)$ *is an optimal solution of (SP), then* $\nu^* = 0$.

(iii) *There is a strictly complementary optimal solution*
$(y^*, w^*, x^*, \tau^*, \nu^* = 0, s^*, z^*, \kappa^*) \in \mathcal{F}_{SP}$,
i.e., $s^*w^* = 0,\ x^*z^* = 0,\ \tau^*\kappa^* = 0,\ s^*+w^* > 0,\ x^*+z^* > 0,$ *and* $\tau^*+\kappa^* > 0$.

THEOREM 2.4. *[27, 32, 35] Let* $(y^*,\ w^*,\ x^*,\ \tau^*,\ \nu^* = 0,\ s^*,\ z^*,\ \kappa^*) \in \mathcal{F}_{sp}$ *be a strictly complementary solution for (SP).*

(i) *If* $\tau^* > 0$ *(so* $\kappa^* = 0$*) then* $(\frac{y^*}{\tau^*}, \frac{w^*}{\tau^*}, \frac{x^*}{\tau^*}, \frac{s^*}{\tau^*}, \frac{z^*}{\tau^*})$ *is a strictly complementary optimal solution to (P) and (D).*

(ii) *If* $\tau^* = 0$ *(so* $\kappa^* > 0$*) then*
if $c^Tx^* < 0$, *then (D) is infeasible;*
if $-b^Ty^* + u^Tw^* < 0$, *then (P) is infeasible;*
if $c^Tx^* < 0$ *and* $-b^Ty^* + u^Tw^* < 0$, *then both (P) and (D) are infeasible.*

2.2.1. Search Directions. Now let us consider the central path for problem (SP) which is defined as the solution set of the system:

$$\begin{array}{rcrcrcrcrcrcl}
 & & & & Ax & - & b\tau & - & r_{\rm P1}\nu & & & & = 0,\\
 & & & - & Fx & + & u\tau & - & r_{\rm P2}\nu & -s & & & = 0,\\
-A^Ty & + & F^Tw & & & + & c\tau & - & r_{\rm D}\nu & & -z & & = 0,\\
b^Ty & - & u^Tw & - & c^Tx & & & - & r_{\rm G}\nu & & & -\kappa & = 0,\\
r_{\rm P1}^Ty & + & r_{\rm P2}^Tw & + & r_{\rm D}^Tx & + & r_{\rm G}\tau & & & & & & = -\beta,\\
 & & & & & & & & & & & Sw & = \mu e,\\
 & & & & & & & & & & & Xz & = \mu e,\\
 & & & & & & & & & & & \tau\kappa & = \mu,
\end{array} \tag{2.5}$$

where for all $\mu > 0$

The Newton system for (2.5) is the following:

$$\begin{pmatrix}
0 & 0 & A & -b & -r_{\rm P1} & 0 & 0 & 0\\
0 & 0 & -F & u & -r_{\rm P2} & -I & 0 & 0\\
-A^T & F^T & 0 & c & -r_{\rm D} & 0 & -I & 0\\
b^T & -u^T & -c^T & 0 & -r_{\rm G} & 0 & 0 & -1\\
r_{\rm P1}^T & r_{\rm P2}^T & r_{\rm D}^T & r_{\rm G}^T & 0 & 0 & 0 & 0\\
0 & S & 0 & 0 & 0 & W & 0 & 0\\
0 & 0 & Z & 0 & 0 & 0 & X & 0\\
0 & 0 & 0 & \kappa & 0 & 0 & 0 & \tau
\end{pmatrix}
\begin{pmatrix}
\Delta y\\ \Delta w\\ \Delta x\\ \Delta\tau\\ \Delta\nu\\ \Delta s\\ \Delta z\\ \Delta\kappa
\end{pmatrix} =
\begin{pmatrix}
0\\ 0\\ 0\\ 0\\ 0\\ \mu e - Sw\\ \mu e - Xz\\ \mu e - \tau\kappa
\end{pmatrix}. \tag{2.6}$$

It is easy to see that when A has full row rank, the coefficient matrix of system (2.6) is nonsingular, thus system (2.6) has a unique solution. The size of the embedding problem (SP) is more than doubled compared with the size of the original LO problem (P). However, computing the search direction (2.6) for the embedding

problem (SP) is only slightly more expensive than for the original problem. Compare to system (2.3), the size of the Newton system (2.6)) increased by three only, from $(m+2mf+2n)\times(m+2mf+2n)$ to $(m+2mf+2n+3)\times(m+2mf+2n+3)$. The small extra effort in solving (SP) brings several important advantages: having a centered initial interior starting point, detecting infeasibility by convergence, and the applicability of any feasible IPMs without degrading theoretical complexity.

3. Self-Regular Proximity Based Feasible IPMs. The new SR-IPMs keep the same structure as classic primal-dual path-following IPMs, but using new proximities (SR-proximities) and new search directions (SR-directions). This new family of IPMs enjoy the best known worst-case complexity [25].

3.1. Self-Regular Proximities. We first give the definition of Self-Regular (SR) functions and present two families of Self-Regular functions that are used in our implementations. More details about Self-Regular (SR) functions are discussed in [25].

DEFINITION 3.1. *The function $\psi(t)\in\mathcal{C}^2:(0,\infty)\to\mathcal{R}_+$ is* self-regular *if it satisfies the following conditions:*

SR1: *$\psi(t)$ is strictly convex with respect to $t>0$ and vanishes at its global minimal point $t=1$, i.e., $\psi(1)=\psi'(1)=0$. Furthermore, there exist positive constants $\nu_2\geq\nu_1>0$ and $p\geq 1, q\geq 1$ such that*

$$\nu_1(t^{p-1}+t^{-1-q})\leq\psi''(t)\leq\nu_2(t^{p-1}+t^{-1-q}),\quad t\in(0,\infty); \tag{3.1}$$

SR2: *For any $t_1,t_2>0$,*

$$\psi(t_1^r t_2^{1-r})\leq r\psi(t_1)+(1-r)\psi(t_2),\quad r\in[0,1]. \tag{3.2}$$

The parameter q and p are called the *barrier degree* and the *growth degree* of $\psi(t)$, respectively.

Two families of SR functions are used in our implementations. The first family, where $\nu_1=\nu_2=1$ and $p\geq 1$, is given by

$$\Upsilon_{p,q}(t)=\begin{cases}\frac{t^{p+1}-1}{p(p+1)}+\frac{p-1}{p}(t-1)-\log t, & \text{if } q=1,\\ \frac{t^{p+1}-1}{p(p+1)}+\frac{t^{1-q}-1}{q(q-1)}+\frac{p-q}{pq}(t-1), & \text{if } q>1.\end{cases} \tag{3.3}$$

The second family, where $\nu_1=1$ and $\nu_2=q$ is given by

$$\Gamma_{p,q}(t)=\frac{t^{p+1}-1}{p+1}+\frac{t^{1-q}-1}{q-1},\quad p\geq 1,\ q>1. \tag{3.4}$$

Note that if $p=q>1$, then $\Gamma_{p,q}(t)=p\Upsilon_{p,q}(t)$.

SR-functions provide a tool to measure the distance between a vector $v\in\mathcal{R}^n_{++}$ and the vector $e=(1,1,\cdots,1)^T\in\mathcal{R}^n$ [25]. An SR-proximity measure $\Psi:\mathcal{R}^n_{++}\to\mathcal{R}_+$ is defined as

$$\Psi(v):=\sum_{i=1}^n\psi(v_i), \tag{3.5}$$

where in this paper we either use $\psi(\cdot)=\Upsilon_{p,q}(\cdot)$ or $\psi(\cdot)=\Gamma_{p,q}(\cdot)$. For notation convenience we also define [25]

$$\nabla\Psi(v)=(\psi'(v_1),\psi'(v_2),\ldots,\psi'(v_n))^T.$$

3.2. Self-Regular Search Directions. To describe the new family of algorithms, we introduce the following notation. Let

$$\begin{aligned} v_{xz} &:= \sqrt{\frac{xz}{\mu}}, \qquad v_{xz}^{-1} := \sqrt{\frac{\mu e}{xz}}, \\ v_{sw} &:= \sqrt{\frac{sw}{\mu}}, \qquad v_{sw}^{-1} := \sqrt{\frac{\mu e}{sw}} \end{aligned} \tag{3.6}$$

denote the vectors whose i^{th} components are $\sqrt{\frac{x_i z_i}{\mu}}$ and $\sqrt{\frac{\mu}{x_i z_i}}$; and j^{th} components are $\sqrt{\frac{s_j w_j}{\mu}}$ and $\sqrt{\frac{\mu}{s_j w_j}}$, respectively. By using this notation, the centrality condition, the last two equations in system (2.2), can be rewritten as $v_{xz} = e$ and $v_{sw} = e$, or $v_{xz}^{-1} = e$ and $v_{sw}^{-1} = e$, or equivalently, in coordinate-wise notation, $v_{xzi} = 1$ for all $i = 1, 2, \cdots, n$ and $v_{swj} = 1$ for all $j = 1, 2, \cdots, m_f$.

The search directions for SR-IPMs suggested by Peng, Roos, and Terlaky [25] is defined by changing the right-hand-side of (2.3) to the following

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ -\mu V_{sw} \nabla \Psi(v_{sw}) \\ -\mu V_{xz} \nabla \Psi(v_{xz}) \end{pmatrix},$$

where V_{sw} and V_{xz} denote the diagonal matrices with diagonal elements v_{sw} and v_{xz}, respectively. Further, $\Psi(v)$ is an SR-proximity function. The resulting search directions are called Self-Regular directions (SR-directions). The iteration bound of large-update IPMs for LO improved by using a SR-proximities and SR-directions as presented in Theorem 3.4.3 of [25]. The iteration complexity of large-update SR-IPMs is

$$O\left(n^{\frac{q+1}{2q}} \log \frac{n}{\varepsilon}\right),$$

better than the previously known $O(n \log \frac{n}{\varepsilon})$ iteration bound. If we choose $p = 2,\ q = \log n$, then the bound for the total number of iterations becomes [25]:

$$O\left(\sqrt{n} \log n \log \frac{n}{\varepsilon}\right).$$

This gives the best known complexity bound for large-update methods at the time we prepare this paper.

The modified Newton system for SR-directions is defined as follow

$$\begin{pmatrix} 0 & 0 & A & -b & -r_{\mathrm{P1}} & 0 & 0 & 0 \\ 0 & 0 & -F & u & -r_{\mathrm{P2}} & -I & 0 & 0 \\ -A^T & F^T & 0 & c & -r_{\mathrm{D}} & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & -r_{\mathrm{G}} & 0 & 0 & -1 \\ r_{\mathrm{P1}}^T & r_{\mathrm{P2}}^T & r_{\mathrm{D}}^T & r_{\mathrm{G}}^T & 0 & 0 & 0 & 0 \\ 0 & S & 0 & 0 & 0 & W & 0 & 0 \\ 0 & 0 & Z & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & \kappa & 0 & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta w \\ \Delta x \\ \Delta \tau \\ \Delta \nu \\ \Delta s \\ \Delta z \\ \Delta \kappa \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ r_{sw} \\ r_{xz} \\ r_{\tau\kappa} \end{pmatrix}, \tag{3.7}$$

where

$$\begin{aligned} r_{sw} &= -\mu V_{sw}\nabla\Psi(v_{sw}), \\ r_{xz} &= -\mu V_{xz}\nabla\Psi(v_{xz}), \\ r_{\tau\kappa} &= -\mu v_{\tau\kappa}\nabla\Psi(v_{\tau\kappa}), \end{aligned} \tag{3.8}$$

and similar as (3.6)

$$v_{\tau\kappa} := \sqrt{\frac{\tau\kappa}{\mu}}, \quad v_{\tau\kappa}^{-1} := \sqrt{\frac{\mu}{\tau\kappa}}.$$

4. From Theory to Computational Practice.

4.1. Computing the Search Directions. Solving system (3.7) is the computationally most involved task in our algorithm. This system has to be solved at each iteration. In this section we derive an efficient way to solve this system. In what follows we give a result that allows us to calculate $\Delta\nu$ in advance, without solving system (3.7). The result holds for both the predictor and the corrector steps.

LEMMA 4.1. *Let $(\Delta y, \Delta w, \Delta x, \Delta\tau, \Delta\nu, \Delta s, \Delta z, \Delta\kappa)$ be the solution of the Newton system (3.7). Then we have*

$$\Delta\nu = \frac{1}{\beta}(e^T r_{sw} + e^T r_{xz} + r_{\tau\kappa}). \tag{4.1}$$

Proof. From (2.4), for any $\alpha \in [0, \alpha_{\max}]$ we have

$$\begin{aligned} (x+\alpha\Delta x)^T(z+\alpha\Delta z) + (s+\alpha\Delta s)^T(w+\alpha\Delta w) + (\tau+\alpha\Delta\tau)(\kappa+\alpha\Delta\kappa) \\ = (\nu+\alpha\Delta\nu)\beta. \end{aligned}$$

Comparing the coefficients of α on both sides, we get

$$\begin{aligned} \Delta\nu\beta = \;& z^T\Delta x + x^T\Delta z + w^T\Delta s + s^T\Delta w + \kappa\Delta\tau + \tau\Delta\kappa \\ = \;& e^T r_{sw} + e^T r_{xz} + r_{\tau\kappa}, \end{aligned} \tag{4.2}$$

that directly implies (4.1). ∎

In order to reduce the computational cost to solve system (3.7), we use the following relations

$$\begin{aligned} \Delta s &= W^{-1}(r_{sw} - S\Delta w), \\ \Delta z &= X^{-1}(r_{xz} - Z\Delta x), \\ \Delta\kappa &= \tau^{-1}(r_{\tau\kappa} - \kappa\Delta\tau), \\ \Delta w &= WS^{-1}r'_{sw} + WS^{-1}F\Delta x - WS^{-1}u\Delta\tau, \end{aligned} \tag{4.3}$$

where $r'_{sw} = r_{\text{P2}}\Delta\nu + W^{-1}r_{sw}$. Then the Newton system (3.7) simplifies to

$$\begin{pmatrix} -D^{-2} & A^T & F^TWS^{-1}u - c \\ A & 0_{m\times m} & -b \\ -u^TWS^{-1}F - c^T & b^T & \dfrac{\kappa}{\tau} + u^TWS^{-1}u \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r''_{xz} \\ r'' \\ r''_{\tau\kappa} \end{pmatrix}, \tag{4.4}$$

where

$$D^2 = (X^{-1}Z + F^TWS^{-1}F)^{-1}, \tag{4.5}$$

and

$$\begin{array}{rl} r'' & = r_{\rm P1}\Delta\nu, \\ r''_{xz} & = r'_{xz} - F^T W S^{-1} r'_{sw}, \\ r''_{\tau\kappa} & = r'_{\tau\kappa} + u^T W S^{-1} r'_{sw}, \\ r'_{xz} & = r_{\rm D}\Delta\nu + X^{-1} r_{xz}, \\ r'_{\tau\kappa} & = r_{\rm G}\Delta\nu + \tau^{-1} r_{\tau\kappa}. \end{array} \tag{4.6}$$

This is the so called augmented system for the self-dual LO problem (SP). System (4.4) can further be reduced to the following so called normal equations form

$$\left[AD^2A^T + \bar{a}\hat{a}^T\right]\Delta y \quad = \hat{r}, \tag{4.7}$$

where

$$\begin{array}{rl} \bar{a} & = \dfrac{a^1}{a^3}, \\ \hat{a} & = -b^T + (c^T + u^T W S^{-1} F) D^2 A^T, \\ \hat{r} & = r_{\rm P1}\Delta\nu - AD^2 r''_{xz} - r'''_{\tau\kappa}\bar{a}, \\ a^1 & = -b - AD^2(c - F^T W S^{-1} u), \\ a^3 & = \dfrac{\kappa}{\tau} + u^T W S^{-1} u + (c^T + u^T W S^{-1} F) D^2 (c - F^T W S^{-1} u), \\ r'''_{\tau\kappa} & = r''_{\tau\kappa} + (c^T + u^T W S^{-1} F) D^2 r''_{xz}. \end{array} \tag{4.8}$$

Finally, $\Delta\tau$ and Δx can be calculated as follows:

$$\begin{array}{rl} \Delta\tau & = \dfrac{1}{a^3}(r'''_{\tau\kappa} + (\hat{a})^T \Delta y), \\ \Delta x & = D^2 r''_{xz} + D^2 A^T \Delta y - D^2(c - F^T W S^{-1} u)\Delta\tau. \end{array} \tag{4.9}$$

Now the key is how to solve system (4.7) efficiently, which is the topic of the following section.

4.1.1. The Normal Equation Approach. Note that the rank-one update in left-hand-side matrix of (4.7) makes the system neither symmetric nor skew-symmetric. However, the computational cost can be reduced by using the Sherman-Morrison formula. Let $P = AD^2A^T$, the solution of system (4.7) can be given as

$$\Delta y = P^{-1}\hat{r} - P^{-1}\bar{a}(I + \hat{a}^T P^{-1}\bar{a})^{-1}\hat{a}^T P^{-1}\hat{r}. \tag{4.10}$$

The following procedure can be used to get Δy:

1. Solve the normal equation system $(AD^2A^T)x_0 = \hat{r}$.
2. Solve $(AD^2A^T)x_1 = \bar{a}$.
3. Finally,

$$\Delta y = \frac{x_0 + x_0\hat{a}^T x_1 - \hat{a}^T x_0 x_1}{1 + \hat{a}^T x_1}.$$

Since D^2 is a positive diagonal matrix, the sparsity pattern of AD^2A^T is independent of the actual value of D, thus the sparse structure of AD^2A^T keeps same at every iteration. Therefore the system $AD^2A^Tx_0 = \hat{r}$ can be solved efficiently by using sparse Cholesky linear system solvers that it only do once symbolic factorization in whole algorithm and once numerical Cholesky factorization at per iteration.

After solving the system (4.7), we can compute $\Delta x, \Delta z, \Delta w, \Delta s, \Delta\tau, \Delta\kappa$ from Δy by using (4.3) and (4.9). However, a drawback of this approach is that if one or more columns of A are dense, the matrix AD^2A^T will be dense. Clearly we need to find an efficient way to reduce the cost of solving such a special structured dense linear system.

4.1.2. Dense Columns. One of the strategies [4] for handing dense columns is to separate the relatively dense columns from sparse ones. For simplicity, let us assume that all the dense columns are the last columns, thus matrix A and D^2 can be partitioned as

$$A = (A_s, A_d), \quad D^2 = \begin{pmatrix} D_s^2 & 0 \\ 0 & D_d^2 \end{pmatrix},$$

where A_s contains the sparse and A_d contains the dense columns of A, and D_s^2 and D_d^2 are reordered accordingly[3]. Now we have

$$AD^2A^T = (A_s, A_d) \begin{pmatrix} D_s^2 & 0 \\ 0 & D_d^2 \end{pmatrix} \begin{pmatrix} A_s^T \\ A_d^T \end{pmatrix} = A_s D_s^2 A_s^T + A_d D_d^2 A_d^T. \tag{4.11}$$

Let $P_s = A_s D_s^2 A_s^T$ and $U = A_d D_d$. The dense matrix AD^2A^T can be split as

$$AD^2A^T = P_s + UU^T,$$

where P_s is the product of the sparse columns and U is the collection of dense columns scaled by the corresponding diagonal elements of D. Matrix U usually has a low rank. Then the system $\left[AD^2A^T + \bar{a}\hat{a}^T\right]\Delta y = \hat{r}$ changes to

$$(P_s + RS^T)\Delta y = \hat{r}, \tag{4.12}$$

where $R = [U \ \ \bar{a}]$, $S^T = [U \ \ \hat{a}]^T$. If A_s is nonsingular, then P_s is nonsingular too. In this case, using the Sherman-Morrison formula again, the solution of (4.12) is

$$\Delta y = P_s^{-1}\hat{r} - P_s^{-1}R(I + S^T P_s^{-1} R)^{-1} S^T P_s^{-1}\hat{r}. \tag{4.13}$$

As we discussed before, $P_s^{-1}\hat{r}$ can be calculated by solving the system $P_s x_0 = \hat{r}$. Similarly, we can get $P_s^{-1}R$ by solving the multi-right-hand-side system $P_s X_1 = R$. Because the same Cholesky factors can be used, we only need multi-right-hand-side back substitutions. Since both R and S have low rank, the matrix $I + R^T P_s^{-1} S$ is dense but it is a low dimensional matrix. Now we can get $(I + S^T P_s^{-1} R)^{-1} S^T x_0$ by solving the system

$$(I + S^T P_s^{-1} R) y_0 = S^T x_0. \tag{4.14}$$

In summary, if A_s is nonsingular, then this strategy for dense columns can be described as follows:

[3]The density of a column is defined by the ratio of the nonzero elements and the number of rows of the matrix A. According to the different possible magnitudes of m, we use the different ratios in McIPM, details are discussed in [35].

Strategy for dense columns

1. Separate the relatively dense columns to get
$$AD^2A^T = P_s + UU^T,$$
where P_s is the sparse part and U is the collection of dense columns that has low rank.
2. Then system (4.7) changes to
$$(P_s + RS^T)\Delta y = \hat{r},$$
where $R = [U \;\; \bar{a}]$ and $S = [U \;\; \hat{a}]$.
3. Solve the symmetric sparse systems
$$P_s x_0 = \hat{r} \;\text{ and }\; P_s X_1 = R.$$
4. Solve the dense system $(I + S^T X_1)y_0 = S^T x_0$.
5. Finally, the solution is $\Delta y = x_0 - X_1 y_0$.

Thus the computational cost will not increase too much comparing the case when we work only with the sparse matrix P. The total cost for the Newton steps of the embedding algorithm with dense columns are one Cholesky decomposition, $O(m^3)$, number of dense column plus one back substitutions with triangular matrices, $O(m^2)$, constant number of addition and multiplication of vectors,$O(n)$.

The assumption in this algorithm is that P_s must be nonsingular. However, after separating the dense columns, the sparse part A_s might be singular, and then P_s is singular too. As a consequence, we cannot apply the above algorithm directly. To handle this case efficiently, a clever trick of K.D. Andersen [4] is needed.

4.1.3. Dealing With Singularity. To ensure the non-singularity of the sparse matrix P_s, a sparse subsidiary matrix H that makes $P_s + HH^T$ nonsingular is needed. We can construct H by applying Gussian Elimination on the matrix A_s to get the upper triangular matrix $\tilde{A}_s$. If A_s is a singular matrix, then there is at least one zero row in its row echelon form $\tilde{A}_s$. Let $i_1, i_2, \ldots, i_{m_1}$, $m_1 < m$, be the indices of the zero rows in $\tilde{A}_s$ and
$$H = (h_{i_1}, h_{i_2}, \ldots, h_{i_{m_1}}),$$
where h_k is the k-th unit vector. We can modify the system (4.12) as follows:
$$[(P_s + HH^T) + (-HH^T + RS^T)]\Delta y = \hat{r}. \tag{4.15}$$
where $P_s + HH^T$ is nonsingular and can be constructed by:
$$P_s + HH^T = [A_s D_s \; H][A_s D_s \; H]^T.$$

Let $\bar{P}_s := P_s + HH^T$. Then $\bar{P}_s$ is a sparse symmetric positive definite matrix. System (4.12) becomes
$$(\bar{P}_s + \bar{R}\bar{S}^T)\Delta y = \hat{r}, \tag{4.16}$$
where $\bar{R} = [-H \;\; R]$, $\bar{S} = [H \;\; S]$. The price we have to pay is that the number of columns of the matrices $\bar{R} = [-H \;\; R]$, and $\bar{S}$ are increased by the number of columns of the matrix H comparing with matrices R and S, and hence the dimension of the dense system (4.14) increases accordingly.

4.1.4. The Augmented System Approach. Another strategy for solving the Newton system (3.7) when matrix A contains dense column is the so called augmented system approach. In this method, instead the normal equation (4.7) we solve the augmented system (4.4). It can be rewritten as

$$\begin{pmatrix} \mathcal{A} & \bar{b} \\ \hat{b}^T & d \end{pmatrix} \begin{pmatrix} \Delta\xi \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r_{xy} \\ r''_{\tau\kappa} \end{pmatrix}, \tag{4.17}$$

where

$$\begin{aligned} \mathcal{A} &= \begin{pmatrix} -D^{-2} & A^T \\ A & 0_{m\times m} \end{pmatrix} \\ \bar{b} &= \begin{pmatrix} F^T S^{-1} W u - c \\ -b \end{pmatrix} \\ \hat{b}^T &= (-u^T S^{-1} W F - c^T,\ b^T) \\ d &= u^T S^{-1} W u + \frac{\kappa}{\tau} \\ r_{xy} &= \begin{pmatrix} r''_{xz} \\ r'' \end{pmatrix} \\ \Delta\xi &= \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}. \end{aligned}$$

After eliminating

$$\Delta\tau = \frac{(r''_{\tau\kappa} - \hat{b}^T \Delta\xi)}{d}, \tag{4.18}$$

system (4.17) reduces to

$$(\mathcal{A} - \frac{1}{d}\,\bar{b}\,\hat{b}^T)\Delta\xi = r_{xy} - \frac{r''_{\tau\kappa}}{d}\,\bar{b}. \tag{4.19}$$

The coefficient matrix of the system (4.19) is a rank-one update of the symmetric indefinite matrix $\mathcal{A}$. Thus we can solve system (4.19) efficiently by rank one update, just as we did for system (4.7) based on solving system $\mathcal{A}\Delta\xi = \text{rhs}$. Since the diagonal elements in matrix $\mathcal{A}$ are non-positive, LDL^T factorizations have to be used instead of Cholesky factorization. After solving system (4.17), we can compute $\Delta\tau$ by (4.18) and compute $\Delta s, \Delta z, \Delta\kappa, \Delta w$ from Δx, Δy, and $\Delta\tau$ by using (4.3).

4.2. SR-Predictor-Corrector Directions. Since the Newton system (3.7) has to be solved at each iteration, one need to compute the Cholesky factorization of the symmetric positive definite matrix at each iteration. The factorization phase is computationally more expensive than the back-solve phase. Therefore, we can allow several back-solves of each iterations if these solves help to reduce the total number of interior point iterations and thus the number of factorizations. Mehrotra's [22] predictor-corrector method is one of these approaches. This method has two components: an adaptive choice of the centering parameter, and the computation of a second-order approximation to the central path.

SR-predictor-corrector IPMs have the same two components as Methrotra's algorithm, but with SR-corrector search directions. The first step of the predictor-corrector strategy is to compute the predictor (affine scaling) direction. The predictor direction can be obtained by solving the Newton equation system (3.7) with the

following right-hand-side

$$r_{sw} = -Sw; \quad r_{xz} = -Xz; \quad r_{\tau\kappa} = -\tau\kappa. \tag{4.20}$$

After the predictor direction, $(\Delta_a y,\ \Delta_a w,\ \Delta_a x,\ \Delta_a \tau,\ \Delta_a \nu,\ \Delta_a s,\ \Delta_a z,\ \Delta_a \kappa)$ is computed, and the maximum feasible step size α_a along this direction is determined, the predicted complementarity gap is calculated as

$$\begin{aligned} g_a = (x + \alpha_a \Delta_a x)^T (z + \alpha_a \Delta_a z) + (s + \alpha_a \Delta_a s)^T (w + \alpha_a \Delta_a w) \\ + (\tau + \alpha_a \Delta_a \tau)(\kappa + \alpha_a \Delta_a \kappa). \end{aligned} \tag{4.21}$$

Then the barrier parameter μ is estimated by the following heuristics: let $\sigma = \left(\frac{g_a}{g}\right)^2$, where $g = x^T z + s^T w + \tau\kappa$ and then let targeted centering parameter be defined as

$$\mu = \sigma \frac{g_a}{n + m_f + 1}. \tag{4.22}$$

Next, the second-order component of the predictor step is computed by solving system (3.7) with the right-hand-side

$$\begin{array}{rl} r_{sw} & = -\mu V_{sw} \nabla\Psi(v_{sw}) - \Delta_a S \Delta_a w, \\ r_{xz} & = -\mu V_{xz} \nabla\Psi(v_{xz}) - \Delta_a X \Delta_a z, \\ r_{\tau\kappa} & = -\mu v_{\tau\kappa} \nabla\Psi(v_{\tau\kappa}) - \Delta_a \tau \Delta_a \kappa, \end{array} \tag{4.23}$$

where μ is given by (4.22), $\Delta_a S$ and $\Delta_a X$ are diagonal matrix with diagonal elements $\Delta_a s_i$ and $\Delta_a x_i$, respectively. Observe that the first component of the corrector direction is based on SR-directions.

4.3. The Maximum Step Length. After get the direction $(\Delta y,\ \Delta w,\ \Delta x,\ \Delta\tau,\ \Delta\nu,\ \Delta s,\ \Delta z,\ \Delta\kappa)$, We need to calculate the maximum acceptable feasible step length α as follows to ensure that the next point will stay in the interior of the feasible range:

$$\begin{array}{rl} \alpha_x^{\max} & = \min_j \left\{ \frac{-x_j}{\Delta x_j} : \Delta x_j < 0, j = 1, \ldots, n \right\}, \\ \alpha_z^{\max} & = \min_j \left\{ \frac{-z_j}{\Delta z_j} : \Delta z_i < 0, i = 1, \ldots, n \right\}, \\ \alpha_w^{\max} & = \min_i \left\{ \frac{-w_i}{\Delta w_i} : \Delta w_j < 0, j = 1, \ldots, m_f \right\}, \\ \alpha_s^{\max} & = \min_i \left\{ \frac{-s_i}{\Delta s_i} : \Delta s_i < 0, i = 1, \ldots, m_f \right\}, \\ \alpha_{\tau\kappa}^{\max} & = \min \left\{ \frac{-\tau}{\Delta\tau},\ \ \frac{-\kappa}{\Delta\kappa} : \Delta\tau < 0, \Delta\kappa < 0 \right\}; \end{array} \tag{4.24}$$

$$\begin{array}{rl} \alpha^{\max} & = \min \left\{ \alpha_x^{\max},\ \alpha_z^{\max},\ \alpha_w^{\max},\ \alpha_s^{\max},\ \alpha_{\tau\kappa}^{\max} \right\}, \\ \alpha & = \min \left\{ \rho \alpha^{\max},\ \ 1 \right\}; \end{array} \tag{4.25}$$

where ρ is a damping parameter to ensure that the next point is not on the boundary[4].

[4]Additional line search can be used to find the points in a certain neighborhood to ensure the next point is not too far away from the center path.

4.4. Stopping Criteria. Another important issue is when to terminate the algorithm and how to give the solution or infeasibility certificate based on the information obtained from the last iteration. Clearly, the algorithm cannot be terminated before a feasible optimal solution of the homogeneous model has been obtained. Feasibility for the embedding model is preserved during the algorithm, therefore we define the following measures:

tol_emd **:** The tolerance for the duality gap of the problem (SP).

tol_ori **:** The tolerance for the duality gap and feasibility for the original problem.

tol_kt **:** Due to the embedding structure, Theorems 2.3 and 2.4 say that one of τ and κ must be zero and the other one must be positive. So *tol_kt* is the tolerance to determine which one is zero in the solution of (SP).

tol_end **:** The end tolerance of the gap of the embedding problem. Due to numerical reasons, this is the smallest value we want to reach for the duality gap of the embedding model (SP).

Define

$$\begin{aligned} fea\_p &= \frac{\|Ax - b\tau\| + \|Fx + s - u\tau\|}{\tau + \|x\|}, \\ fea\_d &= \frac{\|A^T y + z - c\tau - F^T w\|}{\tau + \|z\|}, \\ gap\_ori &= \frac{\|c^T - b^T y + u^T w\|}{\tau + \|b^T y - u^T w\|}, \end{aligned} \tag{4.26}$$

and

$$error = \max\{\ fea\_p,\ fea\_d,\ gap\_ori\ \}.$$

Due to the limitation of machine precision, we cannot reach the precision that theory requires for warranted conclusions. Thus we introduce firm optimal, non-firm optimal, firm infeasible, and non-firm infeasible outcomes. The algorithm will terminate with firm optimal if $error < tol\_ori$ and $\kappa < \tau * tol\_kt$. In this case a solution

$$(x^*, z^*, y^*, w^*, s^*) = \frac{(x^k, z^k, y^k, w^k, s^k)}{\tau^k}$$

is reported to be the optimal solution for (P) and (D). The algorithm will also be terminated if $\mu < tol\_emd$ and $\tau < \kappa * tol\_kt$. In this case a feasible solution to the homogeneous model with a small τ has been computed. Therefore the problem is primal or dual infeasible. Moreover, the algorithm is terminated if $\mu < tol\_end$, and it reports the result up to a certain precision. Figure 4.1 presents the complete structure of our stopping strategy.

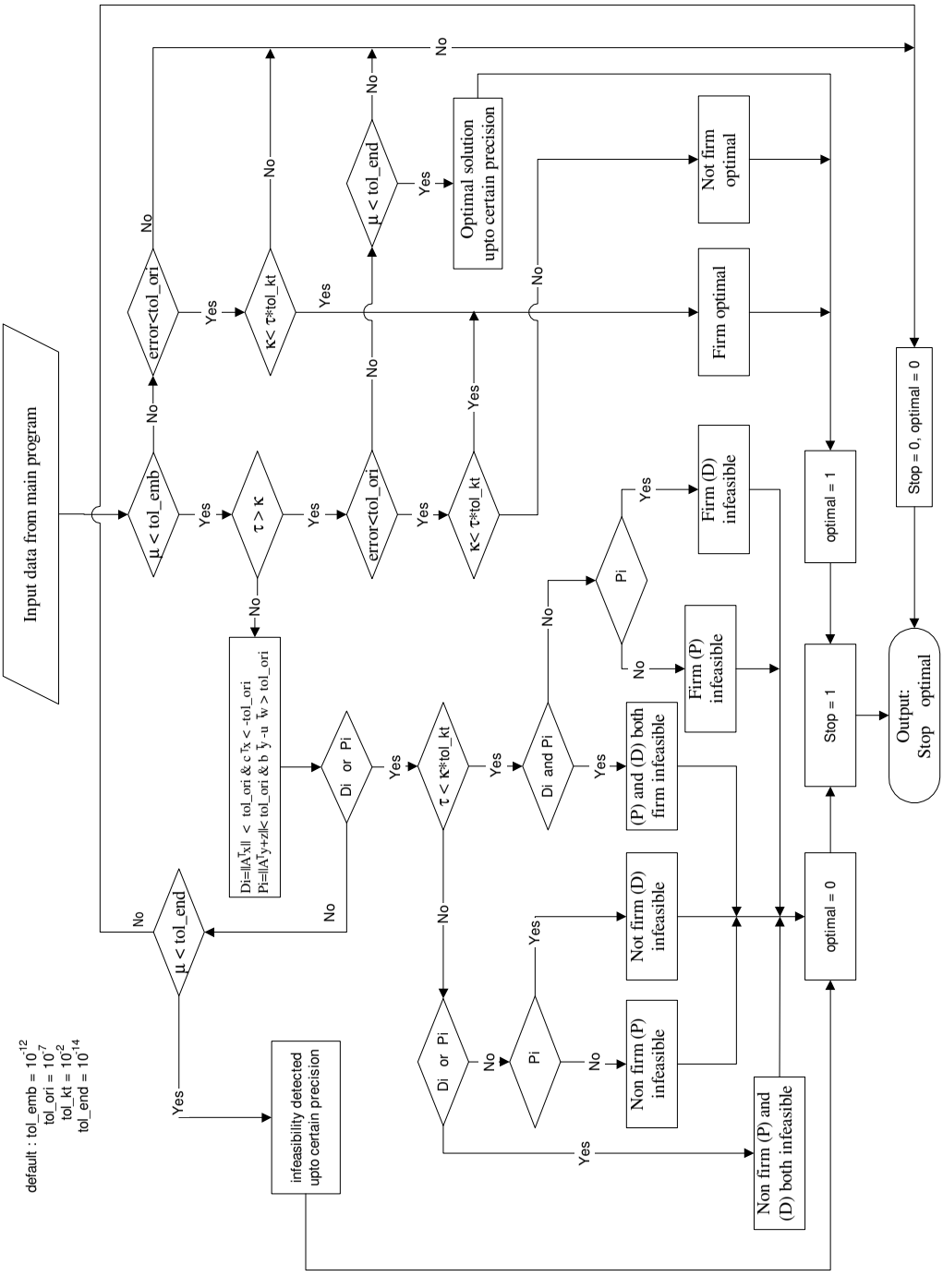

FIG. 4.1. *Flow chart of our stopping criteria.*

4.5. Algorithm. In our implementation, we use a SR-based IPMs with the self-dual embedding model for initialization and infeasibility detection. Further, the predictor-corrector strategy for updating the centering parameter and calculating the search direction is used. In summary, the algorithm can be stated as follows:

SR-IPMs with Predictor-Corrector and Embedding Strategies

Input:
A SR-proximity function $\Psi(v)$
An accuracy parameter $\varepsilon > 0$ and damping parameter $\rho < 1$;
$y^0 = 0, w^0 = e, x^0 = e, \tau^0 = 1, \nu^0 = 1, s^0 = e, z^0 = e, \kappa^0 = 1$.
begin
Set up embedding model (SP) with initial point;
$y := y^0, w := w^0, x := x^0, \tau := \tau^0, \nu := \nu^0, s := s^0, z := z^0, \kappa := \kappa^0$.
while $\beta\nu > \varepsilon$ **do**
begin
Get the affine scaling direction by solving (3.7) with RHS(4.20);
Compute the step size $\alpha_a = \alpha^{\max}$ by (4.25);
Get target μ by (4.21) and (4.22);
Get the corrector direction by solving (3.7) with RHS (4.23);
Compute a step size α from (4.25) and proper line search strategy[5];
Update the current point
$y := y + \alpha\Delta y;\ w := w + \alpha\Delta w;\ x := x + \alpha\Delta x;\ \tau := \tau + \alpha\Delta\tau;$
$\nu := \nu + \alpha\Delta\nu;\ s := s + \alpha\Delta s;\ z := z + \alpha\Delta z;\ \kappa := \kappa + \alpha\Delta\kappa.$
end
Check stop criteria by using the strategy in Figure 4.1.
end.

In this algorithm, one have to choose a SR-proximity function. We will see in Section 6 that the choice of SR-proximity functions considerably influences the computational performance of SR-IPMs.

4.6. Dynamic Self-Regular IPMs. In our algorithm, due to the use of the embedding model, the initial point can be chosen well centered, and even on the central path. In this case the SR-direction coincides with the classical Newton direction. Thus, we use the classical Newton direction at beginning of the algorithm. Later, if a point gets too close to the boundary, then we refer to a SR-direction, because the SR-direction has a stronger barrier property.

In the implementation, we use $q = 1$ in the SR-direction at the beginning of the iteration process, and then dynamically increase q if the maximum step size $\alpha^{\max}$ given by (4.25) is very small, e.g., $\alpha^{\max} < 0.05$. The search direction will change when we use a SR-proximity function with $q > 1$, which will lead to a better centering direction. For simplicity of discussion, we will focus on a special SR-proximity function $\psi(t) = \Gamma_{1,3}(t)$ given by (3.4). When the point is fixed, we can cast the proximity function in (3.8) as a function of μ, e.g.,

$$\Psi(v_{xz}) := \Phi(x, z, \mu) := \frac{x^T z}{2\mu} - n + 2\mu x^{-T} z^{-1}, \tag{4.27}$$

[5]Details are discussed in Section 4.6

where v_{xz} is defined as (3.6). Let

$$\mu_g = \frac{x^T z + s^T w + \tau\kappa}{n + m_f + 1}$$

be the current duality gap and

$$\mu_h = \frac{n + m_f + 1}{x^{-T} z^{-1} + s^{-T} w^{-1} + \tau^{-1}\kappa^{-1}},$$

where $x^{-T} z^{-1} = \sum_{i=1}^{n} (x_i z_i)^{-1}$, and $s^{-T} w^{-1}$ is defined analogously.

From Proposition 2.1 and 2.2 in [26], the proximity function (4.27) has the same value at μ_g and μ_h, $\Phi(x, z, \mu_g) = \Phi(x, z, \mu_h)$, and has a global minimizer at

$$\mu^* = \sqrt{\mu_g \mu_h}. \tag{4.28}$$

In this case, when $\alpha^{\max} < 0.05$ with the classical predictor-corrector direction $q = 1$, we increase q to 3 and take a SR-direction with $\mu = \mu^*$. An inexact froward tracking line search is needed here to make sure that the next iterate is in a certain neighborhood of the central path. This leads to a point close to the central path. If the step size $\alpha^{\max}$ (4.25) is good and $q = 1$, then we keep $q = 1$ and use "backtracking" strategy to ensure that the next point is not too close to the boundary. The froward tracking and 'backtracking" strategies are discussed in Section 4.6.1 and 4.6.2.

Combining this dynamic SR-update strategy with the SR-predictor-corrector algorithm, that we discussed in Section 4.5, we obtain our so-called Dynamic SR-IPM.

Dynamic Self-Regular IPMs

Input:
A family of SR-proximity function $\Gamma_{p,q}(\cdot)$ and parameter max_q;
An accuracy parameter $\varepsilon > 0$ and damping parameter $\rho < 1$;
$y^0 = 0, w^0 = e, x^0 = e, \tau^0 = 1, \nu^0 = 1, s^0 = e, z^0 = e, \kappa^0 = 1$;
$p = 1, q = 1$; (begin with classical Newton direction).

begin
Set up embedding model (SP) with initial point;
$y = y^0, w = w^0, x = x^0, \tau = \tau^0, \nu = \nu^0, s = s^0, z = z^0, \kappa = \kappa^0$.
while $\beta\nu > \varepsilon$ **do**
begin
Get the affine scaling direction by solving (3.7) with RHS (4.20);
Compute the step size $\alpha_a = \alpha^{\max}$ by (4.25);
Get target μ by (4.21) and (4.22);
Get the corrector direction by solving (3.7) with RHS (4.23);
Compute a step size α from (4.25).
while $\alpha \leq 0.05$ and $q <$max_q **do**
begin
Increase q, compute μ^* by (4.28);
Get the new direction with μ^*;
Compute a step size α from (4.25);
Forward tracking.
end
Back tracking;

Set $q := 1$;
Update the current point:
$y := y + \alpha\Delta y;\ w := w + \alpha\Delta w;\ x := x + \alpha\Delta x;\ \tau := \tau + \alpha\Delta\tau;$
$\nu := \nu + \alpha\Delta\nu;\ s := s + \alpha\Delta s;\ z := z + \alpha\Delta z;\ \kappa := \kappa + \alpha\Delta\kappa.$
end
Check stop criteria by using the strategy in Figure 4.1.
end.

See [26] for details about the dynamic SR-IPM. Implementation details of this algorithm are discussed in Section 5.

4.6.1. Froward Tracking. Line search is used for the centering direction with higher $q > 1$ values, to ensure that the iterate gets closer to the central path. This is done through minimizing $\frac{\mu_g}{\mu_h}$, see Lemma 2.3 in [26]. We make an inexact line search, which is called froward tracking to find approximately the closest point to the central path. The line search starts with forward tracking from step size $\alpha = 0.5\alpha^{\max}$, where $\alpha^{\max}$ is calculated by (4.25). If $\frac{\mu_g}{\mu_h} \leq \gamma_0$, where γ_0 is a constant larger than 10, then we accept this point, otherwise we try $\alpha = 0.6\alpha^{\max}$, $\alpha = 0.7\alpha^{\max}$, $\alpha = 0.8\alpha^{\max}$, and $\alpha = 0.9\alpha^{\max}$ in this order, until condition $\frac{\mu_g}{\mu_h} \leq \gamma_0$ is satisfied, or we stop at the point where the $\frac{\mu_g}{\mu_h}$ ratio increases. After a successful higher order step the value of q is reset to 1, and the algorithm proceeds with the next iteration.

4.6.2. Backtracking. With the classical direction, $q = 1$, and a "good" step size, backtracking is used to keep the next point in a certain neighborhood of the central path. Let $v^k = (\frac{Xz}{\mu_g}, \frac{Sw}{\mu_g}, \frac{\tau\kappa}{\mu_g})$ be an iterate in our SR-IPM.

Let us further denote

$$\tilde{v}_s^k := \text{sort}(v^k)$$

and

$$\ell := \min(n, 10\lfloor \log_{10} n \rfloor),$$

where $\text{sort}(v)$ sorts the elements of v in ascending order.

We define the neighborhood of the central path as

$$\mathcal{N} = \{v \mid (w, x, \tau, s, z, \kappa) > 0,\ \tilde{v}_1 > \varepsilon_1 \text{ or } (\tilde{v}_1 > \varepsilon_0 \text{ and } \tilde{v}_\ell > \varepsilon_2)\}, \tag{4.29}$$

where $\varepsilon_0 = 1.e-3,\ \varepsilon_1 = 1.e-2,\ \varepsilon_2 = 1.e-1$ are our default values. Obviously, if $v \in \mathcal{N}$, then $v > 0$. Moreover $v \in \mathcal{N}$ implies that the corresponding point is not too close to the boundary. If $\tilde{v}_1 > \varepsilon_1$, then all coordinates of v are bigger than or equal to ε_1, while in the second case a limited number of relatively smaller coordinates are allowed, while most of the coordinates are much larger.

We start with the step size α from (4.25) and cut back the step size by the fractions [0.9975, 0.95, 0.90, 0.75, 0.50] until the new iterate v^k is in the neighborhood $\mathcal{N}$ or $\alpha = 0.5$.

5. Implementation Issues. McIPM was developed on an IBM RS-6000-44p-270 workstation, with four processors, under the AIX 4.3 operating system and MATLAB 6.1 environment. Our implementation is written in M-files and MEX-files. In McIPM, the routine tasks, mostly matrix and vector operations, are done by MATLAB's M-files. We utilize WSMP [11], the Watson Sparse Matrix Package, as our large linear sparse system solver. Since we do not have the source code of WSMP, the

mass data communication between MATLAB and WSMP turns out to be a major issue. One creative way to solve this problem is running two communicating processes in parallel. A slave process that makes calls to WSMP runs in the background waiting for a 'command' and relevant data (for WSMP input) from the master process. The slave process can be started from MATLAB by making one call in a proper position to a piece of C code that forks this process. The MATLAB code then calls a wrapper code (instead of calling WSMP directly), which passes the data to the slave process, waits to get the results back, and returns them to MATLAB. After finishing a request, the slave process goes back to waiting for the next command to perform some WSMP tasks. Interprocess communication technologies, such as data synchronization, messages passing, and shared memory, are used in McIPM. The details of the design and implementation of the data transfers are discussed in the thesis [35]. The data communication codes are MEX-files generated from C programs by the IBM C/C++ compiler xlc.

We also have developed another version of McIPM, the so called McIPMf, that uses a sparse Cholesky factorization package (version 0.3) developed by E. Ng and B. Peyton at Oak Ridge National Laboratory (ORNL), and a multiple minimum-degree ordering package by J. Liu at University of Waterloo which is included in the ORNL [17]. The following discussion focuses on the package that is implemented based on the sparse matrix package WSMP with the normal equation approach.

In McIPM, LO problems are solved by a standard sequence of procedures: reading data, preprocessing, SR-IPM solver, and postprocessing. LIPSOL's preprocessing and postprocessing subroutines are used in McIPM. The preprocessing function *preprocessing.m* performs a number of tasks, such as checking obvious infeasibility, eliminating fixed variables, zero rows and zero columns from the matrix A, solving equations with one variable, and shifting nonzero lower bounds to zero. The postprocessing function *postprocessing.m* recovers the original problem and provides a solution for the original problem once SR-IPM solves the preprocessed problem.

The kernel part of the McIPM, SR-IPM solver, is based on the algorithm that we have discussed in Sections 4.6 with normal equation approach. The most important part of the implementation of our algorithm is to solve the system $Px = \text{RHS}$ where P is the symmetric positive definite matrix defined by (4.12). We use a sparse Cholesky factorization tool to solve this system efficiently. In practice Cholesky factorization involves three stages. First, a symmetric ordering of the rows and columns of P is chosen in an attempt to minimize the fill-in's, the new nonzero elements created during factorization. Since in IPMs the sparsity pattern of P remains the same at each iteration, so ordering and symbolic factorization are performed only once. Figure 5.1 gives a complete flow chart of this Dynamic SR-IPM.

During implementation and testing several issues needed special attention. For example: what to do if the sparse linear system solver does not give an accurate solution; how to chose a good universal starting point; when and how to terminate the program; how to find a better step length at each iteration than what is given by a fixed fraction of the maximum step length. Some of those issues have been partially addressed in Section 4.1, more details of handling those problem are discussed in [35].

6. Computational Experiences. In this section, we present our computational results. Our numerical experiments were performed on an IBM RS/6000 44P Model 270 Workstation. It is a symmetric multiprocessor (SMP) workstation with four processors and 8GB memory.

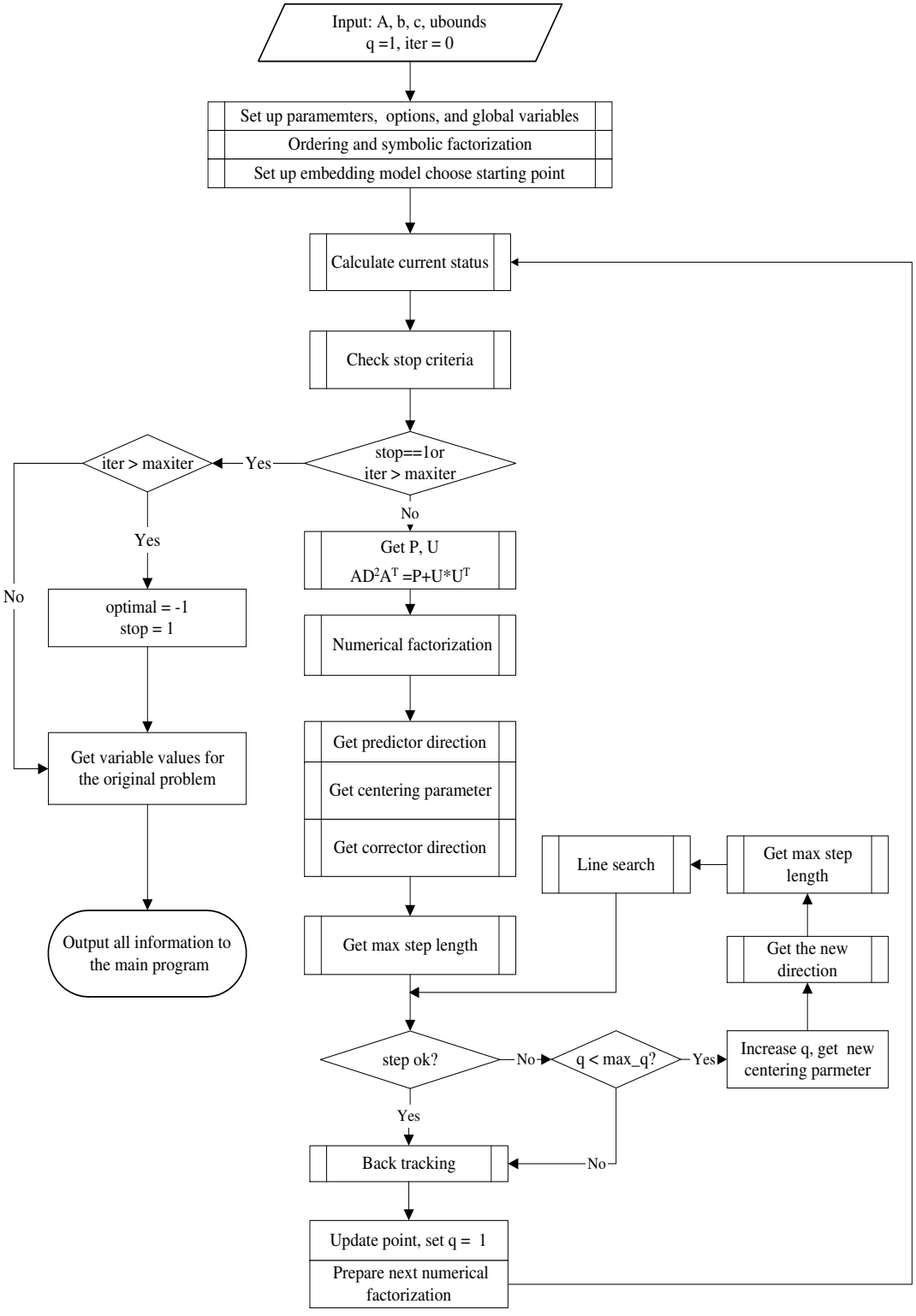

FIG. 5.1. *Dynamic SR-IPMs.*

The test problems used in our experiments come from the Netlib[6] set of LO problems. As of February 2003, the collection consists of three test sets:

Netlib-Standard: The standard test set contains 95 problems, all having primal-dual optimal solutions.

Netlib-Infeasible: The infeasible test set contains 28 problems that are either primal or dual infeasible.

Netlib-Kennington: The Kennington test set contains 16 large scale problems arising from military aircraft applications.

All problems in the Netlib-Standard test set are solved. Our computational results are presented, and compared with the widely used IPM solvers LIPSOL and OSL, in Tables 7.1 to 7.5. Ten columns are given in each tables: the problem name, the number of rows and columns in the matrix A after preprocessing, the number of iterations required for convergence, and the CPU seconds for our solver McIPM, respectively. The column "Digits" shows how many digits agree with the standard solution[7]. For other solvers, LIPSOL and OSL, only the iteration numbers and the number of matching "Digits" are shown. The results demonstrate that McIPM is a stable, robust, and competitive IPM solver, it provides high precision solutions for all of the problems. For some problems, e.g., *dfl001, pilotnov, woodlp, and woodw*, McIPM provides much better performance than LIPSOL and OSL.

There are 28 problems in the Netlib-infeasible set as demonstrated in Table 7.4. Two of them, *gran* and *woodinfe*, were solved in the preprocessing stage. McIPM reported a weak optimal state for problem *cplex2*. LIPSOL reported as optimal like other interior point solvers. OSL gave the correct result since it crosses over to the simplex method at the last stage. The details of our results on *cplex2*, including primal infeasibility, dual infeasibility, and duality gap are reported in Table 7.6. In this table, we also list results from other pure IPM solvers, e.g., MOSEK[8] and PCx[9]. For the remaining 25 problems, we get an infeasibility certificate for all of them, while LIPSOL gives a firm infeasibility certificate only for 12 problems. OSL has a very good preprocessor and solves 13 problems at preprocessing that shows the importance of developing an efficient preprocessing routine. On the other hand, there are 5 problems where OSL reaches the maximum number of iterations and fails to detect infeasibility. McIPM solves all problems of the Netlib-Kennington set with reasonable iteration numbers and high precision. The results are shown in Table 7.5. LIPSOL failed to solve *ken-18* and *osa-60* due to MPS reader error.

We have done test on other problems as well. Those include *baxter, l30, data, rail582, and world*, the results are shown in Table 7.8. For those problems, we present the primal objective value instead the number of matching digits, since there are no standard solution published for comparisons. One can see from Table 7.8, that only for problem *l30* McIPM needs more iterations than LIPSOL, but McIPM reached a smaller objective value. McIPM is able to solve all those problems very efficiently.

In Section 4.1.2, we mentioned the strategy to deal with dense columns in the coefficient matrix A. Table 7.7 illustrates the computational performance with different non-zero/column-length ratios that we used when solving problem *fit2p*. Problem *fit2p* has 3000 rows and 13525 columns with 60748 non-zeroes. The density of a column is defined by the column's non-zero ratio: the number of nonzero elements

[6]http://www-fp.mcs.anl.gov/otc/Guide/TestProblems/LPtest/index.html

[7]http://www-fp.mcs.anl.gov/otc/Guide/TestProblems/LPtest/netlib/README.html.

[8]http://www.mosek.com/

[9]http://www-fp.mcs.anl.gov/otc/Tools/PCx/

divided by the dimension of the column. A column is called dense if its nonzero ratio exceeds a given threshold. The number of dense columns increases in the coefficient matrix A as the threshold value decreases. Table 7.7 shows how solution time, and thus computational performance is effected by the different number of dense columns given by different threshold values.

For evaluating the effect of the choice of Self-Regular search directions and the performance of our Dynamic Self-Regular IPM, we present a set of hard problems in Table 7.9. These problems are identified as difficult ones during our testing, i.e., for all of these problems $q > 1$ was activated in our dynamic SR-IPM algorithm. Table 7.9 gives the number of iterations required to solve these problems for fixed choices of the barrier degree $q = 1, 2, 3, 4$, and our dynamic update strategy in which q is restricted to $q = 1$ and 3. All other parameters were set to the same values as for the results in Table 7.1 to 7.5 and Table 7.8. The last column of this table gives the number of times $q > 1$ was activated during the solution process. Overall, the dynamic version of McIPM has the best result.

Based on these results we may conclude that the dynamic version of our SR-IPM is a very robust algorithm, its performance is always comparable with the classical $q = 1$ variant. There is still great potential for improvement, e.g., to explore the effect of line search when $q > 1$ values are activated and how to find the most efficient strategy for updating the barrier degree q.

7. Conclusions and Future Work. In this paper, we have summarized the fundamental facts about SR-IPMs and discussed implementation details for a family of SR-IPMs that currently form the LO software package: McIPM. Based on our implementation experiences and extensive computational testing, we can claim that SR-IPMs not only have the best theoretical complexity results, they also perform well in practice. SR-IPMs can be implemented efficiently and help to solve difficult LO problems. Our implementation is robust and still has a great potential for improvements.

The encouraging computational results help us to identify items for further improvements. There are still many issues remained in the current version of McIPM, such as numerical stability in calculating the search direction, numerical singularity of the matrix AD^2A^T, sparse Cholesky and Bunch-Parlett factorization with iterative refinement, handling dense columns by the Goldfarb-Scheinberg update, and the pre/post-processing. Any improvement of those are common IPMs implementation issues, may improve our package and make our algorithm and software more efficient and completely independent of third-party software. The most interesting topics for further research in SR-IPMs, what is the best adaptive choice of the self-regular proximity and efficient line-search for SR-search directions. On the other hand, it is also important to consider crossover to simplex and basis-identification procedures. Further, as shown in [25], the SR approach can be extended to more general nonlinear classes. It is worthwhile to consider to extend our package so that it can deal with quadratic optimization, second order conic optimization, and semi-definite optimization problems.

Acknowledgments. We are indebted to E. D. Andersen and Y. Zhang for useful discussions based on their expertise, also to IBM for supporting this project by an FPP grant and A. Gupta for providing the WSMP package and for their contribution to establish the computational power of AdvOL, *the Advanced Optimization Laboratory of McMaster University.*

The first author thanks NSERC (the Natural Sciences and Engineering Research Council of Canada) for a PGS Scholarship, and the Government of Ontario and McMaster University for an OGSST Scholarship, both of those supported her graduate studies.

The research of the authors were partially supported by the grant # RPG 249635-02 and #R.PG 0048923 of NESERC, respectively. This research was also supported by the MITACS project "New Interior Point Methods and Software for Convex Conic-Linear Optimization and Their Application to Solve VLSI Circuit Layout Problems" and CRC, the Canada Researcher Chair Program.

REFERENCES

[1] E.D. Andersen, C. Roos, T. Terlaky, T. Trafalis, and J.P. Warners. The use of low-rank updates in interior-point methods. Technical Report, Delft University of Technology, The Netherlands, 1996.

[2] E.D. Andersen and K.D. Anderson. The Mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm. In J.B.G. Frenk, C. Roos, T. Terlaky and S. Zhang (Editors), *High Performance Optimization*, Kluwer Academic Publishers, Boston, 197–232, 1999.

[3] E.D. Andersen, J. Gondzio, C. Mészáros, and X, Xu. Implementation of Interior Point Methods for Large Scale Linear Programs. In T. Terlaky (Editor), *Interior Point Methods of Mathematical Programming*, volume 5 of *Applied Optimization.* Kluwer Academic Publishers, Boston, 189–252, 1996.

[4] K.D. Andersen. A Modified Schur Complement Method for Handling Dense Columns in Interior-Point Methods for Linear Programming. *ACM Trans. Math. Software*, 22(3), 348–356, 1996.

[5] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, New Jersey, 1963.

[6] Gy. Farkas. *A Fourier-féle mechanikai elv alkalmazásai.* Mathemtikai és Természettudományi Értesítő (in Hungarian) 12, 457-472, 1894.

[7] D.M. Gay. Electronic mail distrubuton of linear programming testing problems. *COAL Newsletter, Mathematical Programming Society*, No. 13, 10-12, 1985.

[8] A.J. Goldman and A.W. Tucker. Theory of linear programming. In H.W. Kuhn and A.W. Tucker. Editors, *Linear Inequalities and Related Systems,* Annals of Mathematical Studies, Princeton, New Jersey, No. 38, 53–97, 1956.

[9] G.H. Golub and C.F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore and London, 2nd Edition, 1989.

[10] O. Güler, D. den Hertog, C. Roos, T. Terlaky, and T. Tsuchiya. Degeneracy in iterior point methods for linear programming: A survey. *Annals of Operations Research*, 46, 107–138, 1993.

[11] A. Gupta. WSMP: Watson Sparse Matrix Package (Part I: direct solution of symmetric sparse systems). Technical Report RC 21886 (98462), IBM T.J. Watson Research Center, Yorktown Heights, NY, 2000. *http://www.cs.umn.edu/~agupta/wsmp.html.*

[12] W.W. Hager. Updating the inverse of a matrix. *SIAM Rev.,* 31(2), 221–239, 1989.

[13] F.S. Hillier and G.J. Lieberman. *Introduction to Operations Research.* McGraw Hill, Boston, 7th Edition, 2001.

[14] N. Karmarkar. A ploynomial-time algorithm for linear programming. *Combinatorica,* 4, 373–395, 1984.

[15] L.G. Khachian. Polynomial algorithmn in linear programming (in Russian). *Doklady Akademiia Nauk SSR*, 224, 1039–1096, 1979. English Translation: *Soviet Mathematics Doklady*, Volumne 20, 191–194, 1979.

[16] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior-point algorithm fo linear programming. In N. Megiddo (Editor), *Progress in Mathematical Programming: Interior-Point Algorithms and Related Methods*, Springer Verlag, Berlin, 29–47, 1989.

[17] J.W.Liu, E.G. Ng and B.W.PEyton. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. & Appl.*, 242–252, 1993.

[18] M. Salahi, T. Terlaky, and G. Zhang. The Complexity of Self-Regular Proximity Based Infeasible IPMs. AdvOl-Report#2003/3, May 2003.

[19] J.W.H. Liu. The multifronttal method for sparse matrix solution: Theory and practice. *SIAM*

Review, 34, 82–109, 1992.

[20] V. Klee and G. Minty. How good is the simplex algorithm? In: O. Shisha (Editor), *Inequalities-III,* Academic Press, New York, 53, 159–175, 1972.

[21] I.J. Lustig, R.E, Marsten, and D.F. Shanno. Interior Point Methods for linear programming problems: Computational State of the Art, *ORSA Journal on Computing,* 6, 1–14, 1994.

[22] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optimization*, 2(4), 575–601, 1992.

[23] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming.* The McGraw-Hill Companies, Inc. New York. International Edition, 1996.

[24] L.L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, New York, Oxford, 1987.

[25] J. Peng, C. Roos, and T. Terlaky. *Self-Regularity: A New Paradigm for Primal-Dual Interior Point Algorithms,* Princeton University Press, Princeton, New Jersey, 2002.

[26] J. Peng and T. Terlaky. A dynamic large-update primal-dual interior-point method for linear optimization. *Optimization Methods and Software*, Vol. 17, No. 6, pp.1077-1104, 2003.

[27] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization. An Interior Approach.* John Wiley and Sons, Chichester, UK, 1997.

[28] T. Terlaky (Editor). *Interior Point Methods of Mathematical Programming,* volume 5 of *Applied Optimization.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[29] T. Terlaky. An easy way to teach interior point methods, *European Journal of Operation Reasearch,* vol. 130, 1, 1–19, 2001.

[30] R. J. Vanderbei. *Linear Programming Foundations and Extensions.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[31] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.* 2, 77-79, 1981.

[32] Y. Ye. *Interior-Point Algorithms, Theory and Analysis.* John Wiley & Sons, Chichester, UK, 1997.

[33] Y. Zhang. Solving large-scale linear programs by interior-point methods under the MATLAB enviroment. *Optimization Methods and Software*, 10, 1-31, 1999.

[34] Y. Zhang. User's Guide to LIPSOL Linear-programming Interior Point Solvers V0.4. *Optimization Methods and Software*, 11 & 12, 385-396, 1999.

[35] X. Zhu. *Implementing the New Self-Regular Proximity Based IPMs.* M.Sc. Thesis, Dept. Computing & Software, McMaster Univ. Aug. 2002.
http://www.cas.mcmaster.ca/~oplab/thesis/zhuMS.pdf

Appendix: Numerical Results. The tables in this section contain our numerical results.

TABLE 7.1
Results for the Netlib-Standard Problems (I).

Problem	Rows	Cols	McIPM		LIPSOL		OSL	
			Iter	Digits	Iter	Digits	Iter	Digits
25fv47	798	1854	27	8	25	11	27	9
80bau3b	2235	11516	40	10	40	9	44	12
adlittle	55	137	14	9	13	11	13	10
afiro	27	51	9	9	8	11	9	10
agg	488	615	21	8	21	11	22	11
agg2	516	758	20	9	18	12	18	11
agg3	516	758	21	9	17	12	17	11
bandm	269	436	17	8	18	10	17	10
beaconfd	148	270	12	9	13	11	9	11
blend	74	114	11	9	12	11	12	11
bnl1	632	1576	33	7	26	7	26	7
bnl2	2268	4430	37	8	33	9	34	11
boeing1	347	722	24	8	21	9	24	9
boeing2	140	279	20	7	19	9	17	10
bore3d	199	300	17	8	18	11	19	12
brandy	149	259	17	9	17	11	17	12
capri	267	476	18	7	20	11	19	11
cycle	1801	3305	38	8	27	8	25	10
czprob	737	3141	32	8	36	11	31	11
d2q06c	2171	5831	42	7	32	7	32	7
d6cube	404	6184	19	10	23	9	21	8
degen2	444	757	12	10	14	11	14	9
degen3	1503	2604	14	9	25	10	19	11
dfl001	6071	12230	46	6	79	7	56	10
e226	220	469	19	10	21	11	21	1
etamacro	357	692	24	8	28	7	34	7
fffff800	501	1005	27	7	26	7	32	7
finnis	492	1014	25	10	30	11	26	6
fit1d	24	1049	24	9	19	11	21	10
fit1p	627	1677	15	8	16	10	16	10
fit2d	25	10524	21	9	7	6	26	11
fit2p	3000	13525	20	9	21	9	?	?
forplan	135	463	30	5	22	6	25	6

Notes:

1) The CPU time for *fit2p* (a problem with dense columns) is obtained by separating the 25 densest columns. If less dense columns are separated, then the CPU time increases significantly. See Table 7.7 for detail.

2) The symbol "?" indicates that the solver failed to solve that problem.

TABLE 7.2
Results for the Netlib-Standard Problems (II).

			McIPM		LIPSOL		OSL	
Problem	Rows	Cols	Iter	Digits	Iter	Digits	Iter	Digits
ganges	1137	1534	20	6	18	6	15	6
gfrd-pnc	600	1144	18	7	21	11	17	10
greenbea	2318	5424	47	3	43	4	48	3
greenbeb	2317	5415	45	8	38	4	59	5
grow15	300	645	18	9	17	11	17	11
grow22	440	946	18	8	19	11	20	11
grow7	140	301	18	8	16	11	17	11
israel	174	316	22	7	23	11	21	9
kb2	43	68	17	9	15	12	17	11
lotfi	151	364	22	6	18	11	15	10
maros	835	1921	30	5	33	11	24	8
maros-r7	3136	9408	17	9	15	11	14	11
modszk1	686	1622	28	6	24	10	24	7
nesm	654	2922	32	6	33	6	38	6
perold	625	1530	44	7	31	6	35	6
pilot	1441	4657	55	6	31	4	34	4
pilotja	924	2044	45	6	31	9	32	8
pilotwe	722	2930	40	6	38	6	44	6
pilot4	402	1173	34	6	30	6	57	6
pilot87	2030	6460	71	5	38	6	23	11
pilotnov	951	2242	26	9	37	6	54	6
recipe	85	177	12	9	9	11	11	11
sc105	105	163	12	7	10	11	10	10
sc205	205	317	12	6	10	9	11	10
sc50a	49	77	10	7	10	11	10	9
sc50b	48	76	9	7	7	8	8	10
scagr25	471	671	16	9	17	11	18	11
scagr7	129	185	14	7	14	7	14	7
scfxm1	322	592	24	8	19	11	16	2
scfxm2	644	1184	24	9	21	12	28	7
scfxm3	966	1776	25	8	21	11	22	9
scorpion	375	453	13	10	15	11	14	11

TABLE 7.3
Results for the Netlib-Standard Problems (III).

			McIPM		LIPSOL		OSL	
Problem	Rows	Cols	Iter	Digits	Iter	Digits	Iter	Digits
scrs8	485	1270	23	5	24	5	22	5
scsd1	77	760	10	8	10	6	10	8
scsd6	147	1350	12	8	12	7	12	12
scsd8	397	2750	10	10	11	10	11	11
sctap1	300	660	18	9	17	12	16	10
sctap2	1090	2500	13	10	19	11	17	10
sctap3	1480	3340	14	10	18	12	18	12
seba	515	1036	23	9	22	11	20	12
share1b	112	248	26	6	22	11	22	9
share2b	96	162	11	7	13	11	13	10
shell	496	1487	23	9	21	12	19	10
ship04l	356	2162	16	9	14	11	17	11
ship04s	268	1414	16	9	14	11	15	11
ship08l	688	4339	19	9	16	11	16	10
ship08s	416	2171	17	10	15	11	16	11
ship12l	838	5329	27	10	18	11	18	11
ship12s	466	2293	22	9	18	12	17	11
sierra	1222	2715	18	9	17	11	19	11
stair	356	538	18	8	16	10	18	3
standata	359	1258	17	10	17	12	15	10
standgub	361	1366	17	10	17	12	15	9
standmps	467	1258	19	11	24	12	21	12
stocfor1	109	157	13	8	17	11	16	10
stocfor2	2157	3045	28	8	22	11	22	11
stocfor3	16675	23541	47	8	34	5	33	5
truss	1000	8806	18	9	19	11	18	10
tuff	292	617	19	8	23	6	19	12
vtpbase	194	325	15	8	28	11	11	10
wood1p	244	2595	15	9	20	7	19	4
woodw	1098	8418	23	10	28	8	25	10

TABLE 7.4
Results for the Netlib-Infeasible Problems.

			McIPM		LIPSOL		OSL	
Problem	Rows	Cols	Iter	Status	Iter	Status	Iter	Status
bgdbg1	348	629	10	P-inf	6	P-inf?	0	P-inf
bgetam	357	692	10	P-inf	6	P-inf?	0	P-inf
bgprtr	20	40	8	P-inf	8	P-inf	26	P-inf
box1	231	261	7	P-inf	4	P-inf	8	P-inf
ceria3d	3576	5224	9	P-inf	?	?	0	P-inf
chemcom	288	744	8	P-inf	6	P-inf	0	P-inf
cplex1	3005	5224	14	P-inf	6	P-inf	4	P-inf
cplex2	224	378	34	optimal?	38	optimal	13	P-inf
ex72a	197	215	8	P-inf	3	P-inf	12	P-inf
ex73a	193	211	7	P-inf	3	P-inf	11	P-inf
forest6	66	131	9	P-inf	10	P-inf	26	P-inf
galenet	8	14	7	P-inf	5	P-inf	0	P-inf
gosh	3718	13625	30	P-inf	16	P-inf?	0	P-inf
gran	2569	2520	0	P-inf	0	P-inf	0	P-inf
greenbeainf	2319	5419	22	P-inf	13	Both-inf?	0	P-inf
itest2	9	13	7	P-inf	5	P-inf	0	P-inf
itest6	11	17	7	P-inf	5	P-inf?	0	P-inf
klein1	54	108	20	P-inf	18	Both-inf?	max	
klein2	477	531	16	P-inf	13	Both-inf?	max	
klein3	994	1082	19	P-inf	16	Both-inf?	max	
mondou2	259	467	14	P-inf	5	P-inf	35	P-inf
pang	357	727	22	P-inf	22	Both-inf?	34	P-inf
pilot4i	402	1173	19	P-inf	16	Both-inf?	0	P-inf
qual	323	459	27	P-inf	40	P-inf	max	
reactor	318	806	20	both-inf	10	P-inf?	0	P-inf
refinery	319	455	13	P-inf	14	P-inf	max	
vol1	323	459	24	P-inf	23	Both-inf?	44	P-inf
woodinfe	36	89	0	P-inf	0	P-inf	0	P-inf

Notes:

1) Iteration number equal to 0 means that infeasibility is detected during preprocessing.

2) McIPM and LIPSOL give an optimal solution for *cplex2*. See Table 7.6 for more details.

3) "max" indicates that the maximum number of iterations is reached. Here max=100.

4) The symbol "?" means that LIPSOL did not produced a firm result.

TABLE 7.5
Results for the Netlib-Kennington Problems.

Problem	Rows	Cols	McIPM		LIPSOL		OSL	
			Iter	Digits	Iter	Digits	Iter	Digits
cre-a	3428	7248	29	10	30	11	35	10
cre-b	7240	77137	35	10	42	11	48	10
cre-c	2986	6411	33	11	30	11	34	10
cre-d	6476	73948	33	10	38	11	51	9
ken-07	1691	2867	17	9	16	11	16	11
ken-11	11548	18203	21	9	22	11	21	11
ken-13	23393	37420	26	10	27	11	25	11
ken-18	105128	154699	45	9	‡		31	11
osa-07	1118	25067	40	10	27	11	24	10
osa-14	2337	54797	38	9	37	10	25	11
osa-30	4350	104374	44	10	36	10	36	11
osa-60	10281	232966	40	10	‡		32	10
pds-02	2788	7551	32	9	29	11	22	11
pds-06	9617	29087	43	8	43	11	34	11
pds-10	16559	48763	59	10	53	11	46	11
pds-20	33875	105728	77	10	67	11	58	11

Notes: The symbol "‡" indicates that the mps reader failed to read those problems.

TABLE 7.6
Results for the Netlib-Infeasible problem cplex2.

	fea-p	fea-d	gap	Primal objective
LIPSOL	1.08e-06	1.56e-13	5.55e-17	6.551371351e-01
McIPM	6.90e-08	2.21e-16	1.52e-09	6.568004817e-01
PCx	2.48e-07	1.45e-12	1.06e-05	6.550883990e-01
MOSEK	1.90e-11	1.60e-13	-7.29e-08	6.569979813e-01

TABLE 7.7
Computational performance with dense columns: Test problem fit2p.

nnz/m	dense col.	time (sec.)
0.20	20	1114.80
0.17	23	303.62
0.1	25	36.87

TABLE 7.8
Results for other Problems

Problem	Rows	Cols	McIPM			LIPSOL	
			Iter	Primal Objective	#SR	Iter	Primal Objective
baxter	24386	30733	49	6.818368E+06	34	57	6.818368E+06
l30	2701	18161	27	9.526627E-01	0	23	9.527772E-01
data	4944	6316	18	4.038465E+02	0	29	4.038465E+02
rail582	582	56097	29	2.097122E+02	15	41	2.097122E+02
world	34081	65875	86	6.913304E+07	63	104	6.913305E+07

Note: #SR: the number of times $q > 1$ was activated during the solution process.

TABLE 7.9
The Performance of different choice of SR-proximity function.

Problem	$q = 1$		$q = 2$		$q = 3$		$q = 4$		Dynamic		
	Iter	dig.	Iter	dig.	Iter	dig.	Iter	dig.	Iter	dig.	#SR
80bau3b	41	8	42	8	48	8	52	8	40	8	6
bnl1	32	7	33	7	36	7	35	7	33	7	8
cycle	39	7	36	8	38	8	39	8	38	7	8
czprob	36	8	31	9	32	9	32	9	34	9	19
d2q06c	43	7	43	7	48	7	55	7	42	7	17
dfl001	46	6	25	1	25	1	47	6	46	6	6
finnis	26	7	27	7	26	7	29	7	25	7	3
greenbea	48	3	52	3	56	3	61	3	47	3	24
maros	31	5	34	5	37	5	44	5	30	4	3
perold	43	7	44	7	45	7	46	7	44	7	12
pilot	75	7	55	7	53	6	52	6	55	7	24
pilotwe	42	6	42	6	41	6	46	6	40	6	2
pilot4	36	6	35	6	39	6	53	6	34	6	4
pilot87	73	5	77	5	75	5	86	5	71	5	38
scfxm2	27	7	27	7	28	7	30	7	26	7	3
shell	24	9	25	9	32	8	28	8	23	9	2
ship04s	17	8	17	10	17	9	19	8	16	8	1
ship08l	19	9	20	10	20	10	22	9	19	9	2
ship08s	17	9	17	9	19	9	20	9	17	9	1
ship12l	28	9	28	9	28	9	31	8	27	8	10
stocfor2	31	7	31	7	30	7	39	7	28	4	6
cplex2	39	0	37	0	36	0	54	0	34	0	14
refinery	14	0	14	0	13	0	17	0	13	0	1
osa-14	40	8	45	8	47	8	45	9	38	8	17
pds-20	79	8	81	9	81	9	100	8	77	8	64
baxter	52	7	51	7	55	8	57	7	49	7	29
rail582*	31	9	29	9	29	9	29	8	29	9	15
world *	85	4	87	4	89	4	118	4	86	4	63

Notes: the maximum number of iterations is set 150.

*: those problems' correct digits are compared with Cplex results.