Math. Program., Ser. A (2008) 112:275–301
DOI 10.1007/s10107-006-0017-0

FULL LENGTH PAPER

A sparse proximal implementation of the LP dual active set algorithm

Timothy A. Davis · William W. Hager

Received: 13 August 2003 / Accepted: 21 June 2006 / Published online: 18 August 2006

Abstract We present an implementation of the LP Dual Active Set Algorithm (LP DASA) based on a quadratic proximal approximation, a strategy for dropping inactive equations from the constraints, and recently developed algorithms for updating a sparse Cholesky factorization after a low-rank change. Although our main focus is linear programming, the first and second-order proximal techniques that we develop are applicable to general concave–convex Lagrangians and to linear equality and inequality constraints. We use Netlib LP test problems to compare our proximal implementation of LP DASA to Simplex and Barrier algorithms as implemented in CPLEX.

Keywords Dual active set algorithm · Linear programming · Simplex method · Barrier method · Interior point method · Equation dropping

Mathematics Subject Classification (2000) 90C05 · 90C06 · 65Y20

This material is based upon work supported by the National Science Foundation under Grant No. 0203270.

T. A. Davis
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, PO Box 116120, FL 32611-6120, USA
e-mail: davis@cise.ufl.edu
URL: http://www.cise.ufl.edu/∼davis

W. W. Hager(✉)
Department of Mathematics, University of Florida,
Gainesville, PO Box 118105, FL 32611-8105, USA
e-mail: hager@math.ufl.edu
URL: http://www.math.ufl.edu/∼hager

1 Introduction

We present an implementation of the LP Dual Active Set Algorithm (LP DASA) [23] for solving linear programming problems. Global convergence is established and comparisons with Simplex and Barrier algorithms, as implemented in CPLEX, are reported. Since our quadratic proximal approach can be applied to general concave–convex Lagrangians, we develop the theory in an abstract setting, and then apply it to the linear programming problem.

Given a function $\mathcal{L} : \mathbf{\Lambda} \times \mathbf{X} \mapsto \mathbb{R}$, where $\mathbf{\Lambda} \subset \mathbb{R}^m$ and $\mathbf{X} \subset \mathbb{R}^n$ are closed and convex, we consider the problem

$$\sup_{\boldsymbol{\lambda} \in \mathbf{\Lambda}} \inf_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}). \tag{1}$$

We refer to $\mathcal{L}$ in (1) as the "Lagrangian," while the "dual function," defined on $\mathbf{\Lambda}$, is given by

$$\mathcal{L}(\boldsymbol{\lambda}) = \inf_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}). \tag{2}$$

Hence, the maximin problem (1) is equivalent to the maximization of the dual function.

Let $\mathbf{c}^\mathsf{T}$ denote the transpose of a vector $\mathbf{c}$. The primal linear program (LP)

$$\min \ \mathbf{c}^\mathsf{T}\mathbf{x} \ \text{ subject to } \ \mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}, \tag{3}$$

where $\mathbf{A}$ is m by n and all vectors are of compatible size, corresponds to the following choices in the dual formulation (1):

$$\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0}\}, \quad \mathbf{\Lambda} = \mathbb{R}^m, \quad \text{and} \tag{4}$$

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) = \mathbf{c}^\mathsf{T}\mathbf{x} + \boldsymbol{\lambda}^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}). \tag{5}$$

Any LP can be written in the canonical form (3) [37]. If the primal problem (3) has a solution $\mathbf{x}^*$, then dual problem (1) has a solution; if $\boldsymbol{\lambda}^*$ is a solution of (1), then $\mathbf{x}^*$ minimizes $\mathcal{L}(\boldsymbol{\lambda}^*, \cdot)$ over $\mathbf{X}$.

Since the dual function for an LP is nonsmooth, we introduce the following "proximal" regularization of the Lagrangian:

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}) = \min_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2, \tag{6}$$

where $\epsilon > 0$ is a scalar and $\mathbf{y} \in \mathbb{R}^n$. First and second-order algorithms for solving (1) are presented. The first-order algorithm corresponds to the following iteration:

$$\boldsymbol{\lambda}_k = \arg \max_{\boldsymbol{\lambda} \in \mathbf{\Lambda}} \mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}_k), \tag{7}$$

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2. \tag{8}$$

The dual active set algorithm (DASA) [20,21,23–25,27] is used to evaluate the maximum in (7). We show that the first-order algorithm amounts to steepest descent along the negative gradient of the function

$$\Omega(\mathbf{y}) = \sup_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}).$$

Moreover, the iteration (7), (8) converges to a saddle point $(\boldsymbol{\lambda}^*, \mathbf{x}^*)$ of the Lagrangian $\mathcal{L}$ and $\mathbf{x}^*$ minimizes Ω over $\mathbb{R}^n$.

The second-order algorithm, based on a quadratic approximation to Ω, converges in a finite number of iterations, while the first-order algorithm (7), (8) converges in the limit, as k tends to infinity. The second-order algorithm requires fewer iterations than the first-order algorithm; however, our current implementation of the first-order algorithm is faster, in terms of CPU time, than the second-order algorithm.

In [23] we develop the LPDASA a finitely convergent algorithm for solving a linear programming problem based on a series of orthogonal projections. In [24, (4.4)] we show that if a proximal regularization is applied to the LP and the resulting quadratic programming problem is solved by the Dual Active Set Algorithm, then the search directions generated by the proximal DASA closely approximate the orthogonal projections generated by LP DASA. In this paper, a complete algorithm based on this proximal approach is developed and analyzed.

Our numerical results use factorization-based sparse matrix techniques [9,10, 12] to solve the linear systems that arise in each step of DASA. In contrast, [24] gives numerical results based on an SSOR iterative solver [22]. This iterative implementation is attractive in cases where the constraint matrix $\mathbf{A}$ is sparse, while the Cholesky factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$ is relatively dense. We saw in [24] that this iterative approach could solve large LPs connected with lower bounds to the quadratic assignment problem, which were intractable for factorization-based simplex, barrier, or dual active set methods.

The iterative approach, however, is not as efficient as a factorization-based approach when both $\mathbf{A}$ and the Cholesky factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$ are sparse. In this case, where the time required to solve a factored linear system may be comparable to the time required to multiply $\mathbf{A}$ by a vector, direct methods are superior to iterative methods. We compare the efficiency of our factorization-based LP DASA scheme to Simplex and Barrier algorithms as implemented in the commercial code CPLEX [3].

Besides developing our proximal LP DASA algorithm, we also develop an important simplification that we call "equation dropping." The basic idea is that as the iterations progress, we seek to identify equations that can be dropped without affecting the solution. We drop these inactive inequalities and solve a simpler problem with fewer constraints. This dynamic strategy for dropping inactive equations is more aggressive than presolve techniques that drop constraints that must always be satisfied.

We remark that another variation of LP DASA, also based on orthogonal projection, appears in [35]. There the bound-constrained least squares problem appearing in the first step of the LP DASA algorithm [21,23] is replaced by an

unconstrained least squares problem. As a result, a nondegeneracy assumption is needed to ensure convergence, while we obtain in [23] convergence without any assumptions on the problem. Previous applications of DASA have been to classes of bound-constrained control problems [2,26,41] and to quadratic network optimization [25].

We briefly compare LP DASA [23] to both simplex and barrier approaches. In the dual active set approach, we start with a guess for the dual multiplier associated with the constraint $\mathbf{Ax} = \mathbf{b}$, and we ascend the dual function, eventually obtaining a subset of the columns of $\mathbf{A}$ that contains $\mathbf{b}$ in its span. Either $\mathbf{b}$ is a nonnegative combination of these columns of $\mathbf{A}$, and we have obtained an optimal solution of the LP, or some of the columns are discarded, and the iteration repeats. Since the algorithm constantly ascends the dual function, the collection of columns obtained in each iteration does not repeat, and convergence is obtained in a finite number of steps. This finite convergence property is similar to that of the Simplex method, where the iterates travel along edges of the feasible set, descending the cost function in a finite number of steps. Unlike the Simplex method, neither rank nor nondegeneracy assumptions are either invoked or facilitate the analysis. In essence, one is able to prove finite convergence without any assumptions. In the Simplex method, typically one constraint is activated and one constraint is deactivated in each iteration. In contrast, many constraints may change in an LP DASA iteration.

In Barrier methods, the iterates move through the relative interior of the feasible region. In LP DASA, the iterates are infeasible, and the algorithm seeks to identify the columns of $\mathbf{A}$ associated with an optimal basis. Each iteration of the Barrier method computes a scaled projection of the cost vector into the null space of $\mathbf{A}$. LP DASA projects the constraint violation vector into the space orthogonal to the "free" columns of $\mathbf{A}$. Consequently, a Barrier method solves symmetric linear systems involving a matrix whose sparsity pattern coincides with that of $\mathbf{AA}^{\mathsf{T}}$; LP DASA solves symmetric linear systems with the matrix $\mathbf{A}_F\mathbf{A}_F^{\mathsf{T}}$, where $\mathbf{A}_F$ is the submatrix of $\mathbf{A}$ corresponding to the current free variables (primal variables not at a bound). In a sparse setting, the LP DASA matrix $\mathbf{A}_F\mathbf{A}_F^{\mathsf{T}}$ is often much more sparse than the barrier matrix associated with $\mathbf{AA}^{\mathsf{T}}$. Thus storage requirements and solution times associated with the LP DASA solves can be less than those for Barrier methods.

Our paper is organized as follows: in Sect. 2 we prove convergence of the first-order algorithm (7), (8), and we present the dual active set algorithm that we use to solve (7). Section 3 explores the relationship between primal and dual iterates, and reveals that the algorithm (7), (8) is a gradient descent method applied to Ω. The second-order algorithm is presented in Sect. 4. In Sect. 5, we develop a strategy for deleting "inactive equations" when using DASA. Numerical comparisons with Simplex and Barrier codes in CPLEX are given in Sect. 6.

1.1 Notation

The following notation is used throughout the paper: The gradient of a real-valued function Ω is a row vector denoted $\nabla\Omega$. For a function of two variables,

say $\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x})$, the gradient with respect $\boldsymbol{\lambda}$ is $\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x})$. The Euclidean norm of a vector $\mathbf{x}$ is denoted $\|\mathbf{x}\|$, while $\mathbf{x}^{\mathsf{T}}$ denotes the transpose of $\mathbf{x}$. The subscript k is often used to denote an iteration number in an algorithm, while x_{kj} stands for the j-th component of the iterate $\mathbf{x}_k$. If $F \subset \{1, 2, \ldots, n\}$, then $\mathbf{A}_F$ is the submatrix of $\mathbf{A}$ associated with columns corresponding to indices in F. The complement of F is denoted F^c. The set of integers F is often associated with components of the primal variable $\mathbf{x}$ that are "free" in the sense that $x_j > 0$. A set B of integers is often associated with bound primal components; that is, components for which $x_j = 0$. Since the "basic variables" in LP are typically free, the bound set B set is essentially the opposite of the "basic variables" in LP.

2 The first-order proximal update and DASA

In this section we observe that the iteration (7), (8) is convergent to a saddle point of $\mathcal{L}$ when one exists, and we review DASA. If $\mathcal{L}(\boldsymbol{\lambda}, \cdot)$ is convex, then the extremand in (6) is strongly convex in $\mathbf{x}$, and the minimizing argument, denoted $\mathbf{x}(\boldsymbol{\lambda})$, is unique. As a consequence of [5, Theorem 2.1], if the derivative $\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x})$ is continuous with respect to $\boldsymbol{\lambda}$ and $\mathbf{x}$, then $\mathcal{L}_\epsilon(\,\cdot\,; \mathbf{y})$ is differentiable and

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}) = \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}(\boldsymbol{\lambda})). \tag{9}$$

If the mixed derivative $\nabla_{\mathbf{x}} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x})$ is continuous, then by [18, Lemma 2.2] or [19, Theorem 4.1], $\mathbf{x}(\boldsymbol{\lambda})$ is Lipschitz continuous in $\boldsymbol{\lambda}$, and by (9), $\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y})$ is Lipschitz continuously differentiable in $\boldsymbol{\lambda}$. Hence, by introducing the ϵ term in (6), we are able to transform a (possibly) nonsmooth extremand in (1) into a smooth function $\mathcal{L}_\epsilon(\cdot; \mathbf{y})$.

We now make a simplifying assumption: the Lagrangian $\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x})$ is strongly concave in $\boldsymbol{\lambda}$. That is, there exists $\delta > 0$, independent of $\mathbf{x}$, such that

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) \leq \mathcal{L}(\boldsymbol{\mu}, \mathbf{x}) + \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\mu}, \mathbf{x})(\boldsymbol{\lambda} - \boldsymbol{\mu}) - \frac{\delta}{2} \|\boldsymbol{\lambda} - \boldsymbol{\mu}\|^2 \tag{10}$$

for each $\boldsymbol{\lambda}, \boldsymbol{\mu} \in \boldsymbol{\Lambda}$. In this case, the maximizer $\boldsymbol{\lambda}_k$ in (7) is unique. Technically, an LP does not satisfy this strong concavity condition. On the other hand, similar to the regularization in $\mathbf{x}$ done in (6), we could augment $\mathcal{L}$ with a small concave term of the form $-\delta \|\boldsymbol{\lambda}\|^2$ in order to satisfy this condition. From a practical viewpoint, this small term is insignificant in the computations. From a theoretical viewpoint, this small term yields a bound on the iterates generated by (7), (8), from which convergence follows. Alternatively, for an LP, a similar bound on the iterates can be achieved by letting $\boldsymbol{\lambda}_k$ be the minimum-norm maximizer in (7).

2.1 Global convergence

Recall that $(\boldsymbol{\lambda}^*, \mathbf{x}^*) \in \boldsymbol{\Lambda} \times \mathbf{X}$ is a saddle point of $\mathcal{L}$ if

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}^*) \leq \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) \leq \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}) \quad \text{for all } \boldsymbol{\lambda} \in \boldsymbol{\Lambda} \text{ and } \mathbf{x} \in \mathbf{X}. \tag{11}$$

The second inequality implies that

$$\mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) = \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}),$$

while the first inequality implies that

$$\mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) \geq \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}).$$

Hence, a saddle point is a solution of (1).

Theorem 1 *Suppose the Lagrangian $\mathcal{L}(\cdot,\cdot)$ is continuously differentiable, $\mathcal{L}(\,\cdot\,,\mathbf{x})$ is uniformly, strongly concave for each fixed $\mathbf{x} \in \mathbf{X}$, and $\mathcal{L}(\boldsymbol{\lambda},\,\cdot\,)$ is convex for each fixed $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. If $\mathcal{L}$ has a saddle point $(\boldsymbol{\lambda}^*, \mathbf{x}^*) \in \boldsymbol{\Lambda} \times \mathbf{X}$, then the $\boldsymbol{\lambda}_k$ generated by* (7), (8) *converge to the maximizer $\boldsymbol{\lambda}^*$ of* (1) *and the associated $\mathbf{y}_k$ converge to a minimizer $\mathbf{y}^*$ of the function*

$$\Phi(\mathbf{x}) = \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}).$$

The pair $(\boldsymbol{\lambda}^*, \mathbf{y}^*) \in \boldsymbol{\Lambda} \times \mathbf{X}$ is a saddle point of $\mathcal{L}$, and for each k, we have

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}_{k+1}; \mathbf{y}_{k+1}) \leq \mathcal{L}_\epsilon(\boldsymbol{\lambda}_k; \mathbf{y}_k) - \frac{\delta}{2}\|\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k\|^2 - \frac{\epsilon}{2}\|\mathbf{y}_{k+1} - \mathbf{y}_k\|^2. \tag{12}$$

Proof We focus on the iterates associated with $k = 1$ and $k = 2$. Putting $\mathbf{x} = \mathbf{y}_2$ in (6) when $\boldsymbol{\lambda} = \boldsymbol{\lambda}_2$ and $\mathbf{y} = \mathbf{y}_2$, and exploiting the uniform, strong concavity of $\mathcal{L}(\,\cdot\,,\mathbf{x})$, gives

$$\begin{aligned}\mathcal{L}_\epsilon(\boldsymbol{\lambda}_2; \mathbf{y}_2) = \min_{\mathbf{x}\in\mathbf{X}} \;\; \mathcal{L}(\boldsymbol{\lambda}_2, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_2\|^2 &\leq \mathcal{L}(\boldsymbol{\lambda}_2, \mathbf{y}_2) \\ &\leq \mathcal{L}(\boldsymbol{\lambda}_1, \mathbf{y}_2) + \nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{\lambda}_1, \mathbf{y}_2)(\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1) - \frac{\delta}{2}\|\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1\|^2. \end{aligned} \tag{13}$$

Also, by the first-order optimality conditions at the maximizer in (7) and by (9), it follows that

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{y}_{k+1})(\boldsymbol{\lambda} - \boldsymbol{\lambda}_k) \leq 0 \tag{14}$$

for all $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. By the definition of $\mathbf{y}_2$,

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}_1; \mathbf{y}_1) = \mathcal{L}(\boldsymbol{\lambda}_1, \mathbf{y}_2) + \frac{\epsilon}{2}\|\mathbf{y}_2 - \mathbf{y}_1\|^2. \tag{15}$$

We combine (13), (14) with $k = 1$, and (15) to obtain

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}_2; \mathbf{y}_2) \leq \mathcal{L}_\epsilon(\boldsymbol{\lambda}_1; \mathbf{y}_1) - \frac{\delta}{2}\|\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1\|^2 - \frac{\epsilon}{2}\|\mathbf{y}_2 - \mathbf{y}_1\|^2. \tag{16}$$

Analogous to (16), the iterates $(\boldsymbol{\lambda}_{k+1}, \mathbf{y}_{k+1})$ and $(\boldsymbol{\lambda}_k, \mathbf{y}_k)$ satisfy (12), which establishes monotone descent.

The function

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2$$

is strongly convex in $\mathbf{x}$ for each fixed $\boldsymbol{\lambda}$ and strongly concave in $\boldsymbol{\lambda}$ for each fixed $\mathbf{x}$. Hence, we have the following identity (see [15, p. 173]):

$$\begin{aligned}\max_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2 &= \min_{\mathbf{x}\in\mathbf{X}} \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2 \\ &= \min_{\mathbf{x}\in\mathbf{X}} \Phi(\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2. \end{aligned} \tag{17}$$

Moreover, by [15, p. 167], if $\boldsymbol{\lambda}_k$ and $\mathbf{y}_{k+1}$ achieve the maximum and the minimum in (17) respectively, then $(\boldsymbol{\lambda}_k, \mathbf{y}_{k+1})$ is a saddle point:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\lambda}, \mathbf{y}_{k+1}) + \frac{\epsilon}{2}\|\mathbf{y}_{k+1} - \mathbf{y}_k\|^2 &\le \mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{y}_{k+1}) + \frac{\epsilon}{2}\|\mathbf{y}_{k+1} - \mathbf{y}_k\|^2 \\ &\le \mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2\end{aligned}$$

for all $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ and $\mathbf{x} \in \mathbf{X}$. It follows that

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2.$$

Hence, the $\mathbf{x}$ achieving the minimum in (17), denoted $\mathbf{y}_{k+1}$ above, coincides with the iterate $\mathbf{y}_{k+1}$ given in (8), and the iteration (7), (8) is equivalent to the proximal point iteration

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{x}\in\mathbf{X}} \Phi(\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}_k\|^2.$$

Since $\mathcal{L}(\boldsymbol{\lambda}, \cdot)$ is convex, Φ is convex; that is, take $\theta \in [0, 1]$ and $\mathbf{x}_1$ and $\mathbf{x}_2 \in \mathbf{X}$ to obtain

$$\begin{aligned}\Phi(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) &= \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}(\boldsymbol{\lambda}, \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) \\ &\le \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \{\theta\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}_1) + (1-\theta)\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}_2)\} \\ &\le \theta\Phi(\mathbf{x}_1) + (1-\theta)\Phi(\mathbf{x}_2). \end{aligned} \tag{18}$$

Due to the saddle point assumption (11), we have (see [15, p. 167])

$$\mathbf{x}^* \in \arg \min_{\mathbf{x}\in\mathbf{X}} \Phi(\mathbf{x}).$$

By the convexity of Φ and the existence of a minimizer, the convergence theory for the proximal point method (see [32,33,36, Theorem 1]) tells us that the $\mathbf{y}_k$ approach a minimizer $\mathbf{y}^*$ of Φ; obviously,

$$\Phi(\mathbf{x}^*) = \Phi(\mathbf{y}^*). \tag{19}$$

By the saddle point assumption (11), we have

$$\Phi(\mathbf{y}^*) = \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{y}^*) \geq \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}^*) \geq \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) = \Phi(\mathbf{x}^*). \tag{20}$$

Since $\Phi(\mathbf{y}^*) = \Phi(\mathbf{x}^*)$, the inequalities in (20) are equalities. Hence,

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{y}^*). \tag{21}$$

By (20) and the saddle point condition (11), it follows that

$$\mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}^*) = \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) \leq \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}) \tag{22}$$

for all $\mathbf{x} \in \mathbf{X}$. Together, (21) and (22) imply that $(\boldsymbol{\lambda}^*, \mathbf{y}^*)$ is a saddle point of $\mathcal{L}$.

We now show that the $\boldsymbol{\lambda}_k$ approach $\boldsymbol{\lambda}^*$. By (21) and the first-order optimality conditions for $\boldsymbol{\lambda}^*$, we have

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}^*)(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) \leq 0 \text{ for all } \boldsymbol{\lambda} \in \boldsymbol{\Lambda}. \tag{23}$$

Adding (23) with $\boldsymbol{\lambda} = \boldsymbol{\lambda}_k$ to (14) with $\boldsymbol{\lambda} = \boldsymbol{\lambda}^*$ gives

$$\Big(\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}^*) - \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{y}_{k+1})\Big)(\boldsymbol{\lambda}_k - \boldsymbol{\lambda}^*) \leq 0. \tag{24}$$

By the uniform strong convexity assumption, we have

$$\Big(\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}_{k+1}) - \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_k, \mathbf{y}_{k+1})\Big)(\boldsymbol{\lambda}_k - \boldsymbol{\lambda}^*) \geq \delta \|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}^*\|^2. \tag{25}$$

Combining (24) and (25) yields the following:

$$\delta \|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}^*\|^2 \leq \Big(\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}_{k+1}) - \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}^*)\Big)(\boldsymbol{\lambda}_k - \boldsymbol{\lambda}^*)$$

We apply the Schwarz inequality to obtain

$$\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}^*\| \leq \|\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}_{k+1}) - \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{y}^*)\|/\delta.$$

Since the $\mathbf{y}_k$ approach $\mathbf{y}^*$, the continuity of $\nabla_{\boldsymbol{\lambda}} \mathcal{L}$ implies that the $\boldsymbol{\lambda}_k$ approach $\boldsymbol{\lambda}^*$. □

Remark 1 Although ϵ is independent of k in the statement of Theorem 1, it could depend on k without any changes in the analysis. When ϵ depends on k, (12) becomes:

$$\mathcal{L}_{\epsilon_{k+1}}(\boldsymbol{\lambda}_{k+1};\mathbf{y}_{k+1}) \leq \mathcal{L}_{\epsilon_k}(\boldsymbol{\lambda}_k;\mathbf{y}_k) - \frac{\delta}{2}\|\boldsymbol{\lambda}_{k+1}-\boldsymbol{\lambda}_k\|^2 - \frac{\epsilon_k}{2}\|\mathbf{y}_{k+1}-\mathbf{y}_k\|^2.$$

2.2 Dual active set algorithm

Let us assume that the primal constraint set $\mathbf{X}$ is the nonnegative cone (4). The statement of the dual active set algorithm (DASA) for solving (1), appearing in Algorithm 1, uses the following notation: Given a *bound set* $B \subset \{1,2,\ldots,n\}$, let $\mathbf{x}_B$ be the subvector of $\mathbf{x}$ consisting of those components x_j associated with $j \in B$. We define the functions

$$\mathcal{L}_{B^+}(\boldsymbol{\lambda}) = \min\{\mathcal{L}(\boldsymbol{\lambda},\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n, \mathbf{x}_B \geq \mathbf{0}\}, \tag{26}$$

and

$$\mathcal{L}_{B^0}(\boldsymbol{\lambda}) = \min\{\mathcal{L}(\boldsymbol{\lambda},\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n, \mathbf{x}_B = \mathbf{0}\}. \tag{27}$$

The components of $\mathbf{x}$ in (26) and (27) corresponding to indices in the complement of B are unconstrained during the minimizations. Assuming $\mathcal{L}(\boldsymbol{\lambda},\mathbf{x})$ is strongly convex in $\mathbf{x}$, there exists a unique minimizer $\mathbf{x}(\boldsymbol{\lambda},B)$ in (26) for each choice of $\boldsymbol{\lambda}$ and B. Let $\mathbf{x}(\boldsymbol{\lambda})$ denote $\mathbf{x}(\boldsymbol{\lambda},B)$ in the case $B = \{1,2,\ldots,n\}$.

In Algorithm 1, there is an outer loop indexed by l, and an inner loop indexed by k. In the inner loop, indices are deleted from B_k to obtain B_{k+1}; that is, B_{k+1} is a subset of B_k consisting of those indices for which $x_j(\boldsymbol{\nu}_{k+1},B_k) = 0$. When the inner loop terminates, we reinitialize B_0 by solving the problem

Algorithm 1 Dual Active Set Algorithm

$l = 0$
$\boldsymbol{\lambda}_0$ = starting guess
while $\nabla\mathcal{L}(\boldsymbol{\lambda}_l) \neq \mathbf{0}$
  $\boldsymbol{\nu}_0 = \boldsymbol{\lambda}_l$
  $B_0 = \{j \in [1,n] : x_j(\boldsymbol{\lambda}_l) = 0\}$.
  for $k = 0,1,\ldots$
    $\boldsymbol{\omega}_k =$ arg max $\{\mathcal{L}_{B_k^0}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \mathbb{R}^m\}$
    $\boldsymbol{\nu}_{k+1} =$ arg max $\{\mathcal{L}_{B_k^+}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in [\boldsymbol{\nu}_k,\boldsymbol{\omega}_k]\}$.
    $B_{k+1} = \{j \in B_k : x_j(\boldsymbol{\nu}_{k+1},B_k) = 0\}$
    if $\boldsymbol{\nu}_{k+1} = \boldsymbol{\omega}_k$ break
  end
  $l = l+1$
  $\boldsymbol{\lambda}_l = \boldsymbol{\omega}_k$
end

$$\min_{\mathbf{x}\geq\mathbf{0}} \mathcal{L}(\boldsymbol{\lambda}_l, \mathbf{x})$$

to obtain $\mathbf{x}(\boldsymbol{\lambda}_l)$. Those j for which $x_j(\boldsymbol{\lambda}_l) = 0$ form the new B_0. In the LP setting studied here, or in the networks studied in [25], the indices contained in the final B_k of the subiteration are included in the initial B_0 of the next iteration. Hence, the initialization of B_0 adds indices to the current bound set, while the evaluation of B_{k+1} frees bound indices. The proof [23, Theorem 1] that DASA solves (1) in a finite number of iterations is based on the fact that the subiteration strictly increases the value of the dual function; as a result, the final set B_k generated in the subiteration cannot repeat. Since the sets B_k are chosen from a finite set, convergence occurs in a finite number of steps.

We use DASA to solve (7). We now explain each step of Algorithm 1 in the case where the dual function is the regularized function $\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y})$ associated with the LP (3):

1. In the outer loop of Algorithm 1, the iteration continues until the gradient of the dual function vanishes. As in (9), this gradient can be expressed

$$\nabla \mathcal{L}_\epsilon(\boldsymbol{\lambda}) = \mathbf{b} - \mathbf{A}\mathbf{x}(\boldsymbol{\lambda}), \tag{28}$$

   where $\mathbf{x}(\boldsymbol{\lambda})$ achieves the minimum in (6) corresponding to the LP-dual function (5):

$$x_j(\boldsymbol{\lambda}) = \max\{0, z_j(\boldsymbol{\lambda})\}, \quad \mathbf{z}(\boldsymbol{\lambda}) = \mathbf{y} - \frac{\mathbf{c} - \mathbf{A}^\mathsf{T}\boldsymbol{\lambda}}{\epsilon}. \tag{29}$$

2. Before the inner loop, B_0 is initialized to be the indices j for which $x_j(\boldsymbol{\lambda}_l) = 0$, where $\mathbf{x}(\boldsymbol{\lambda})$ is given in (29).
3. In the first step of the inner iteration, we maximize $\mathcal{L}_{B_k^0}$. This amounts to solving the problem

$$\max_{\boldsymbol{\lambda}} \ \min\{\mathbf{c}^\mathsf{T}\mathbf{x} + \boldsymbol{\lambda}^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2 : \mathbf{x} \in \mathbb{R}^n, \mathbf{x}_{B_k} = \mathbf{0}\}. \tag{30}$$

   If $F = B_k^c$ is the complement of B_k, then the maximun problem (30) is equivalent to the following equations:

$$(\mathbf{A}_F\mathbf{A}_F^\mathsf{T})\boldsymbol{\lambda} = \mathbf{A}_F\mathbf{c}_F + \epsilon(\mathbf{b} - \mathbf{A}_F\mathbf{y}_F), \quad \text{and} \tag{31}$$

$$\mathbf{x}_F = \mathbf{y}_F - \frac{1}{\epsilon}(\mathbf{c}_F - \mathbf{A}_F^\mathsf{T}\boldsymbol{\lambda}), \tag{32}$$

   where $\mathbf{A}_F$ denotes the submatrix of $\mathbf{A}$ associated with columns corresponding to indices in F. The solution $\boldsymbol{\lambda}$ of (31) is the $\boldsymbol{\omega}_k$ of Algorithm 1. The $\mathbf{x}_F$ given in (32) is the associated primal minimizer in (30).

4. The computation of $\boldsymbol{v}_{k+1}$ represents a line search in which the piecewise quadratic $\mathcal{L}_{B_k^+}(\boldsymbol{\lambda})$ is maximized along the line segment connecting the current subiterate $\boldsymbol{v}_k$ and the maximizer $\boldsymbol{\omega}_k$ (the solution of (31)). For any $\boldsymbol{\lambda}$ and B, we have

$$\mathcal{L}_{B^+}(\boldsymbol{\lambda}) = \mathbf{c}^\mathsf{T}\mathbf{x} + \boldsymbol{\lambda}^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2, \tag{33}$$

where

$$x_j(\boldsymbol{\lambda}) = \begin{cases} \max\{0, z_j(\boldsymbol{\lambda})\} & \text{if } j \in F = B^c, \\ 0 & \text{if } j \in B, \end{cases} \tag{34}$$

As $\boldsymbol{\lambda}$ moves along the line segment $[\boldsymbol{v}_k, \boldsymbol{\omega}_k]$, $\mathbf{z}(\boldsymbol{\lambda})$ in (29) changes linearly with $\boldsymbol{\lambda}$ and $x_j(\boldsymbol{\lambda})$ in (34) changes in a piecewise linear fashion. Hence, $\mathcal{L}_{B^+}$ in (33) is a convex, piecewise quadratic function.

5. By the definition of x_j in (34), B_{k+1} is obtained by pruning from B_k those indices $j \in B_k$ for which $x_j(\boldsymbol{v}_{k+1}) > 0$.
6. If $x_j^k(\boldsymbol{v}_{k+1}) = 0$ for all $j \in B_k$, then the subiteration terminates and the set B_0 is reinitialized to be the final B_k of the subiteration plus all indices $j \in F = B_k^c$ for which $z_j(\boldsymbol{\omega}_k) \le 0$.

Numerical details connected with our DASA implementation are given in Sect. 6. Briefly, we maintain a Cholesky factorization of the matrix $\mathbf{A}_F\mathbf{A}_F^\mathsf{T} + \sigma\mathbf{I}$, which is updated as the free set F changes. Here the columns of $\mathbf{A}$ are normalized to be unit vectors and σ, a multiple of the machine epsilon, is chosen to ensure numerical stability in the evaluation of the Cholesky factorization. In our numerical experiments, σ was around 200 times the machine epsilon. Sparse numerical linear algebra techniques for updating this factorization are developed in [9,10,12]. Typically, several indices are dropped from B_k in each subiteration (step 5 above); also, the initial set B_0 (constructed in step 6) often has many more indices than the final set B_k of the previous iteration. Hence, multiple-rank updates of $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$ are used in step 5, and multiple-rank downdates are used in step 6. If the rank of the update or the downdate is large enough, then it can be more efficient to completely reform the matrix $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$ and compute its Cholesky factorization directly.

Remark 2 It was pointed out by a referee that (31), (32) is equivalent to the following symmetric indefinite system

$$\begin{pmatrix} \sqrt{\epsilon}\mathbf{I} & \mathbf{A}_F^\mathsf{T} \\ \mathbf{A}_F & \mathbf{0} \end{pmatrix} \begin{pmatrix} -\sqrt{\epsilon}\mathbf{x}_F \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_F - \epsilon\mathbf{y}_F \\ -\sqrt{\epsilon}\mathbf{b} \end{pmatrix}. \tag{35}$$

If the $\mathbf{0}$ block is replaced by $-\sigma\mathbf{I}$, then the matrix in (35) becomes quasi-definite, as defined by Vanderbei [40]. Some theory in papers by Golub and Van Loan [17] and by Gill et al. [16] is used by Saunders [38, Result 3, p. 94]) to show that an indefinite $\mathbf{LDL}^\mathsf{T}$ factorization exists for any symmetric permutation. Moreover, the factorization is *sufficiently stable* if $\sigma \gg u\|\mathbf{A}_F\|^2$ (u = unit roundoff

or machine epsilon), in the sense that it behaves like a stable factorization on a matrix with condition number $\|\mathbf{A}_F\|^2/\sigma$. For sparse $\mathbf{A}$, the Cholesky factor of this matrix can be much sparser than that of $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$. In particular, when $\mathbf{A}_F$ has a dense column, the matrix $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}+\sigma\mathbf{I}$ and its factorization become dense; on the other hand, the augmented matrix (35) has sparsity comparable to that of $\mathbf{A}_F$, and with a suitable permutation of rows and columns, the Cholesky factorization of the augmented matrix may be relatively sparse. Hence, the augmented matrix (35) would allow for the treatment of dense columns in a natural way, without having to resort to Sherman–Morrison–Woodbury techniques.

3 Primal and dual relationships

Let Ω be defined by

$$\Omega(\mathbf{y}) = \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \inf_{\mathbf{x}\in\mathbf{X}} \left\{\mathcal{L}(\boldsymbol{\lambda},\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x}-\mathbf{y}\|^2\right\} = \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}_\epsilon(\boldsymbol{\lambda};\mathbf{y}). \tag{36}$$

Observe that the first step (7) of the first-order proximal algorithm is equivalent to evaluating $\Omega(\mathbf{y}_k)$. Under the assumptions of Theorem 1, it follows from [5, Theorem 2.1] that the function Ω is differentiable and

$$\nabla\Omega(\mathbf{y}) = \epsilon(\mathbf{y}-\mathbf{x}(\mathbf{y})), \tag{37}$$

where $\mathbf{x}(\mathbf{y})$ is the minimizing $\mathbf{x}$ in (36) corresponding to the maximizing $\boldsymbol{\lambda}$. Thus the update (8) corresponds to a step of length $1/\epsilon$ along the negative gradient of Ω:

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \left(\frac{1}{\epsilon}\right)\nabla\Omega(\mathbf{y}_k).$$

In Theorem 1, we show that

$$\Omega(\mathbf{y}_{k+1}) \le \Omega(\mathbf{y}_k) - \frac{\delta}{2}\|\boldsymbol{\lambda}_{k+1}-\boldsymbol{\lambda}_k\|^2 - \frac{\epsilon}{2}\|\mathbf{y}_{k+1}-\mathbf{y}_k\|^2.$$

Hence, each iteration of (7) and (8) generates descent for the function Ω; we can think of (7), (8) as a first-order gradient algorithm applied to Ω with a fixed stepsize of length $1/\epsilon$. The following theorem shows that the first-order algorithm, under the hypotheses of Theorem 1, converges to a minimizer of Ω.

Theorem 2 *If $\mathcal{L}(\,\cdot\,,\mathbf{x})$ is concave for each fixed $\mathbf{x}\in\mathbf{X}$ and $\mathcal{L}(\boldsymbol{\lambda},\,\cdot\,)$ is convex for each fixed $\boldsymbol{\lambda}\in\boldsymbol{\Lambda}$, where $\mathbf{X}\subset\mathbb{R}^n$ and $\boldsymbol{\Lambda}\subset\mathbb{R}^m$ are convex, then Ω is convex on $\mathbb{R}^n$. If $\mathcal{L}$ has a saddle point $(\boldsymbol{\lambda}^*,\mathbf{x}^*)\in\boldsymbol{\Lambda}\times\mathbf{X}$, then $(\boldsymbol{\lambda}^*,\mathbf{x}^*)$ is a saddle point of $\mathcal{L}_\epsilon$, $\mathbf{x}^*$ minimizes Ω over $\mathbb{R}^n$, and*

$$\Omega(\mathbf{x}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*,\mathbf{x}^*) = \mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}^*) = \mathcal{L}(\boldsymbol{\lambda}^*). \tag{38}$$

If $\mathbf{y}^* \in \mathbf{X}$ *is any minimizer of* Ω*, then* $(\boldsymbol{\lambda}^*, \mathbf{y}^*)$ *is a saddle point of* $\mathcal{L}_\epsilon$*, and* $\mathbf{y}^*$ *minimizes* $\mathcal{L}(\boldsymbol{\lambda}^*, \cdot)$ *over* $\mathbf{X}$*.*

Proof For any fixed $\mathbf{x} \in \mathbf{X}$ and $\mathbf{y} \in \mathbb{R}^n$, the function

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{y}) = \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2$$

is concave in $\boldsymbol{\lambda}$ by assumption. Similar to the analysis (18), but with convexity replaced by concavity, $\mathcal{L}_\epsilon(\,\cdot\,; \mathbf{y})$ is concave. We now show that for any fixed $\boldsymbol{\lambda}$, $\mathcal{L}_\epsilon(\boldsymbol{\lambda};\,\cdot\,)$ is convex. First, observe that the function $\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{y})$, for fixed $\boldsymbol{\lambda}$, is convex with respect to the pair $(\mathbf{x}, \mathbf{y})$ because the Lagrangian $\mathcal{L}(\boldsymbol{\lambda},\,\cdot\,)$ is convex by assumption while the term $\|\mathbf{x} - \mathbf{y}\|^2$ is convex with respect to the pair $(\mathbf{x}, \mathbf{y})$. Given any $\mathbf{x}_i, \mathbf{y}_i \in \mathbf{X}$, $i = 1, 2$, and $\theta \in [0, 1]$, the convexity of $\mathcal{L}(\boldsymbol{\lambda}, \cdot, \cdot)$ yields

$$\begin{aligned}\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \theta\mathbf{y}_1 + (1-\theta)\mathbf{y}_2) &= \min_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}, \theta\mathbf{y}_1 + (1-\theta)\mathbf{y}_2) \\ &\le \mathcal{L}(\boldsymbol{\lambda}, \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2, \theta\mathbf{y}_1 + (1-\theta)\mathbf{y}_2) \\ &\le \theta\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}_1, \mathbf{y}_1) + (1-\theta)\mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}_2, \mathbf{y}_2).\end{aligned}$$

Minimizing over $\mathbf{x}_1$ and $\mathbf{x}_2 \in \mathbf{X}$ gives

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \theta\mathbf{y}_1 + (1-\theta)\mathbf{y}_2) \le \theta\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}_1) + (1-\theta)\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}_2).$$

Hence, $\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \cdot)$ is convex on $\mathbb{R}^n$. As in (18), Ω is convex on $\mathbb{R}^n$.

For any $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, the first inequality in the saddle point condition (11) and the definition of $\mathcal{L}_\epsilon$ give

$$\mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) \ge \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}^*) \ge \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{x}^*\|^2 = \mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{x}^*). \tag{39}$$

From the second inequality in (11), we have, for any $\mathbf{y} \in \mathbb{R}^n$,

$$\mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}^*) = \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}) \le \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}^*, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{y} - \mathbf{x}\|^2 = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*; \mathbf{y}). \tag{40}$$

Combining (39) and (40) gives

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{x}^*) \le \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*; \mathbf{x}^*) \le \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*; \mathbf{y}) \text{ for all } \boldsymbol{\lambda} \in \boldsymbol{\Lambda},\ \mathbf{y} \in \mathbb{R}^n. \tag{41}$$

Since Ω is given by a maximum over $\boldsymbol{\lambda}$ in (36), we have, for any $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ and $\mathbf{y} \in \mathbb{R}^n$,

$$\Omega(\mathbf{y}) \ge \inf_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2 \ge \mathcal{L}(\boldsymbol{\lambda}). \tag{42}$$

Again, by the saddle point assumption (11),

$$\mathcal{L}(\boldsymbol{\lambda}^*) = \max\{\mathcal{L}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \boldsymbol{\Lambda}\}.$$

Combining this with (42) gives

$$\Omega(\mathbf{y}) \geq \mathcal{L}(\boldsymbol{\lambda}^*) \text{ for all } \mathbf{y} \in \mathbb{R}^n. \tag{43}$$

By the definition of Ω, the new saddle point condition (41), and (43), we have

$$\begin{aligned}\Omega(\mathbf{x}^*) &= \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}_\epsilon(\boldsymbol{\lambda};\mathbf{x}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}^*) = \inf_{\mathbf{y}\in\mathbf{X}} \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{y}) \\ &= \inf_{\mathbf{x},\mathbf{y}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x}-\mathbf{y}\|^2 = \mathcal{L}(\boldsymbol{\lambda}^*) \leq \Omega(\mathbf{y}).\end{aligned} \tag{44}$$

Hence, $\mathbf{x}^*$ minimizes Ω over $\mathbb{R}^n$. Putting $\mathbf{y} = \mathbf{x}^*$ in (44), we conclude that the inequalities are equalities, and $\Omega(\mathbf{x}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}^*) = \mathcal{L}(\boldsymbol{\lambda}^*)$. Also, $\mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}^*) = \mathcal{L}(\boldsymbol{\lambda}^*)$ since $(\boldsymbol{\lambda}^*,\mathbf{x}^*)$ is a saddle point of $\mathcal{L}$. This establishes (38).

If $\mathbf{y}^*$ minimizes Ω over $\mathbf{X}$, then by the saddle point condition (41) and by (44), we have

$$\Omega(\mathbf{y}^*) = \sup_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathcal{L}_\epsilon(\boldsymbol{\lambda};\mathbf{y}^*) \geq \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{y}^*) \geq \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}^*) = \Omega(\mathbf{x}^*). \tag{45}$$

Since $\Omega(\mathbf{y}^*) = \Omega(\mathbf{x}^*)$, the inequalities in (45) are equalities, and $\mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{y}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}^*)$. Again, by the saddle point condition (41), we have

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{y}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}^*) \leq \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}) \tag{46}$$

for all $\mathbf{x} \in \mathbf{X}$. Together, (45) and (46) imply that $(\boldsymbol{\lambda}^*,\mathbf{y}^*)$ is a saddle point of $\mathcal{L}_\epsilon$.

Combining (38) with (46) gives

$$\mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{x}^*) = \mathcal{L}_\epsilon(\boldsymbol{\lambda}^*;\mathbf{y}^*) = \min_{\mathbf{x}\in\mathbf{X}} \mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x}-\mathbf{y}^*\|^2. \tag{47}$$

If the minimum in (47) is attained at $\bar{\mathbf{x}}$, then

$$\mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}^*) = \mathcal{L}(\boldsymbol{\lambda}^*,\bar{\mathbf{x}}) + \frac{\epsilon}{2}\|\bar{\mathbf{x}}-\mathbf{y}^*\|^2. \tag{48}$$

If $\bar{\mathbf{x}} \neq \mathbf{y}^*$, then (48) implies that $\mathcal{L}(\boldsymbol{\lambda}^*,\bar{\mathbf{x}}) < \mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}^*)$. This violates the saddle point condition (11). Hence, the minimum in (47) is achieved at $\mathbf{x} = \mathbf{y}^*$, and $\mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{x}^*) = \mathcal{L}(\boldsymbol{\lambda}^*,\mathbf{y}^*)$. Since $\mathbf{x}^*$ minimizes $\mathcal{L}(\boldsymbol{\lambda}^*,\cdot)$ over $\mathbf{X}$, it follows that $\mathbf{y}^*$ minimizes $\mathcal{L}(\boldsymbol{\lambda}^*,\cdot)$ over $\mathbf{X}$. □

4 A second-order proximal update

In this section we present a second-order algorithm, first in a general setting, and then in the LP setting. An attractive feature of the second-order algorithm is that it converges in a finite number of iterations, while Theorem 1 yields convergence of the first-order algorithm in the limit, as k tends to ∞. On the other

Algorithm 2 Second-order proximal update for the case where $\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0}\}$

$l = 0$
$\mathbf{y}_0$ = starting guess
while $\nabla\Omega(\mathbf{y}_l) \neq \mathbf{0}$
  $\boldsymbol{\eta}_0 = \mathbf{y}_l$
  $B_0 = \{j \in [1,n] : x_j(\mathbf{y}_l) = 0\}$.
  for $k = 0, 1, \ldots$
    Case 1. $x_j(\bar{\boldsymbol{\eta}}, B_k) \geq 0$ for all $j \in B_k^c$ where
      $\bar{\boldsymbol{\eta}} =$ arg min $\{\Omega_{B_k}(\mathbf{y}) : \mathbf{y} \in \mathbb{R}^n\}$
      set $\boldsymbol{\eta}_{k+1} = \bar{\boldsymbol{\eta}}$ and break
    Case 2. for some $j \in B_k^c$ and $\bar{\boldsymbol{\eta}} \in \mathbb{R}^n$,
      $\Omega_{B_k}(\bar{\boldsymbol{\eta}}) < \Omega_{B_k}(\boldsymbol{\eta}_k)$ and $x_j(\bar{\boldsymbol{\eta}}, B_k) < 0$
      set $\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}(\alpha)$ where
        $\boldsymbol{\eta}(\alpha) = \boldsymbol{\eta}_k + \alpha(\bar{\boldsymbol{\eta}} - \boldsymbol{\eta}_k)$
        $\alpha = \min\{\beta \in [0,1] : x_j(\boldsymbol{\eta}(\beta), B_k) = 0 \text{ for some } j \in B_k^c\}$
      set $B_{k+1} = \{j \in [1,n] : x_j(\boldsymbol{\eta}_{k+1}, B_k) = 0\}$
  end
  $l = l + 1$
  $y_{lj} = \max\{0, \bar{\eta}_j\}$, $1 \leq j \leq n$
end

hand, our implementation of the second-order algorithm is currently less efficient than our implementations of the first-order algorithm. Later we explain the reason for slower performance of the second-order algorithm.

We consider the specific case where $\mathbf{X}$ is the nonnegative cone (4) and $\boldsymbol{\Omega} = \mathbb{R}^m$. In the statement of Algorithm 2, we use the following notation: For any $B \subset \{1, 2, \ldots, n\}$, define

$$\Omega_B(\mathbf{y}) = \max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \min_{\mathbf{x}_B = \mathbf{0}} \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2}\|\mathbf{x} - \mathbf{y}\|^2. \tag{49}$$

We let $\mathbf{x}(\mathbf{y}, B)$ denote the $\mathbf{x}$ that achieves the minimum in (49) corresponding to the maximizing $\boldsymbol{\lambda}$, assuming it exists.

Algorithm 2 involves two cases. In Case 1, $\boldsymbol{\eta}_{k+1}$ is an unconstrained minimizer of Ω_{B_k} and the inner iteration terminates. In Case 2, there is no unconstrained minimizer which lies in $\mathbf{X}$. In this case, we take any infeasible point with a smaller function value and perform a line search. For a concave/convex Lagrangian, it can be shown that Algorithm 2 converges in a finite number of iterations.[1]

For an LP, Ω_B is a quadratic. Hence, in Algorithm 2, the iterates are generated by minimizing quadratics, while in Algorithm 1, the iterates are generated by taking of step of length $1/\epsilon$ along $-\nabla\Omega$. For this reason, we regard Algorithm 2 as a second-order algorithm.

For an LP, Cases 1 and 2 greatly simplify.[1] In particular, if the quadratic Ω_{B_k} has a finite lower bound, then in an LP it can be shown that there exists a minimizer $\bar{\boldsymbol{\eta}}$ with $x_j(\bar{\boldsymbol{\eta}}) \geq 0$ for all $j \in B_k^c$. If F denotes B_k^c, then the new iterate can be expressed in the following way:

[1] See the appendix to this paper at http://www.math.ufl.edu/~hager/papers/LPDASA.

$$\boldsymbol{\eta}_{k+1,F} = \boldsymbol{\eta}_{k,F} + \Delta\boldsymbol{\eta},$$

where $\Delta\boldsymbol{\eta}$ is a solution to

$$\min_{\Delta\boldsymbol{\eta}} \|\Delta\boldsymbol{\eta}\| \quad \text{subject to} \quad \mathbf{A}_F\Delta\boldsymbol{\eta} = \mathbf{b} - \mathbf{A}_F\boldsymbol{\eta}_{k,F}.$$

Case 2 corresponds to the situation where the quadratic Ω_{B_k} has no lower bound. The iterates, which move in the null space of the Hessian, can be expressed

$$\boldsymbol{\eta}_{k+1,F} = \boldsymbol{\eta}_{k,F} + \alpha(\mathbf{A}_F^{\mathsf{T}}\mathbf{z} - \mathbf{c}_F),$$

where $\mathbf{z}$ is a solution to the least squares problem

$$\min_{\mathbf{z}} \|\mathbf{A}_F^{\mathsf{T}}\mathbf{z} - \mathbf{c}_F\|,$$

and α is chosen as large as possible, subject to the constraint

$$\boldsymbol{\eta}_{k+1,F} - \frac{1}{\epsilon}(\mathbf{c}_F - \mathbf{A}_F^{\mathsf{T}}\boldsymbol{\lambda}_k) \geq 0.$$

Here $\boldsymbol{\lambda}_k$ is the maximizer in (49) corresponding to $\mathbf{y} = \boldsymbol{\eta}_k$ and $B = B_k$. If α can be chosen infinitely large, then the optimal LP cost is $-\infty$.

In our numerical experiments, the first-order algorithm is more efficient than the second-order algorithm for the following reason: In the inner loop of Algorithm 1, each iteration typically removes many indices from B_k. As a result, fast techniques we have developed [10] for updating a Cholesky factorization after a multiple-rank change can be exploited. In contrast, in the inner loop of Algorithm 2, Case 2, each iteration typically adds one index to B_k. Hence, the less efficient rank 1 techniques in [9] are utilized. Thus, for the second-order algorithm to surpass the first-order algorithm, it appears that a line search must be developed that allows B_k to change more rapidly.

5 Equation dropping

In this section, we focus on a strategy that we call "equation dropping." Although we apply the strategy to linear programs, our strategy is applicable to problems that contain a variable appearing in only one equation; this variable should appear linearly in both the equation and the cost function. An important situation where this occurs is the following: An inequality is converted to an equality by introducing a slack variable. The slack variable appears in precisely one equation, while it is absent from the objective function.

Given an optimal solution $\mathbf{x}$ of the LP, we say that equation i is inactive if there exists a column j of $\mathbf{A}$ that is completely zero except for an element a_{ij} for which $x_j > 0$. In this case, it follows from complementary slackness that at a corresponding solution to the dual problem, $\lambda_i a_{ij} = c_j$. Thus, if the inactive

equations were known, we could hold the associated multipliers $\lambda_i = c_j/a_{ij}$ fixed during the solution of the dual problem, and solve an equivalent reduced problem with the inactive rows removed.

Since we do not know the inactive equations when we start to solve the dual problem, the solution algorithm must dynamically identify the inactive equations. Our approach is roughly the following: Initially, we locate all the column singletons and we compute associated multipliers $\lambda_i = c_j/a_{ij}$. In the line search of Algorithm 1, any component of λ_i that reaches the value c_j/a_{ij}, associated with a column singleton, is held fixed and equation i is dropped from the problem. Thus in the inner loop of DASA, rows are dropped and columns are freed. In the initialization step of DASA, where B_0 is defined, we also check whether any dropped rows should be restored to the problem. Row i is restored if the value of the dual function can be increased with a small change in the previously fixed value of λ_i. These steps are now explained in more detail.

Let $\mathcal{S}$ denote the set of column indices j corresponding to column singletons. That is, for each $j \in \mathcal{S}$, there is precisely one i such that $a_{ij} \neq 0$. Let $\mathcal{I}(j)$ denote the row index associated with column singleton j; hence, $\mathcal{I}^{-1}(i)$ is the set of column singletons in row i. If $j, k \in \mathcal{I}^{-1}(i)$ and $c_j/a_{ij} = c_k/a_{ik}$, then the variables x_j and x_k can be combined together into a single variable [1]. To simplify the exposition, we assume that the LP has been presolved so that $c_j/a_{ij} \neq c_k/a_{ik}$ for all $j, k \in \mathcal{I}^{-1}(i)$.

When column singletons are present, we delete the regularization terms in (6) associated with the set $\mathcal{S}$:

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}) = \min_{\mathbf{x} \in \mathbf{X}} \ \mathcal{L}(\boldsymbol{\lambda}, \mathbf{x}) + \frac{\epsilon}{2} \sum_{j \in \mathcal{S}^c} (x_j - y_j)^2$$

For an LP, the Lagrangian (5) is a separable function of $\boldsymbol{\lambda}$, and we have

$$\mathcal{L}_\epsilon(\boldsymbol{\lambda}; \mathbf{y}) = \sum_{j=1}^{n} \mathcal{L}_j(\boldsymbol{\lambda}),$$

where

$$\mathcal{L}_j(\boldsymbol{\lambda}) = \begin{cases} \min\limits_{x_j \geq 0} \ (c_j - \lambda_i a_{ij}) x_j, \ i = \mathcal{I}(j), & \text{if } j \in \mathcal{S}, \\ \min\limits_{x_j \geq 0} \ (c_j - \sum\limits_{i=1}^{m} a_{ij} \lambda_i) x_j + \dfrac{\epsilon}{2} (x_j - y_j)^2 & \text{otherwise.} \end{cases}$$

If $j \in \mathcal{S}$, then

$$\mathcal{L}_j(\boldsymbol{\lambda}) = \begin{cases} 0 & \text{if } a_{ij}\lambda_i \leq c_j, \\ -\infty & \text{if } a_{ij}\lambda_i > c_j. \end{cases}$$

Since the dual function is being maximized, we should never allow $a_{ij}\lambda_i > c_j$; in the case that $a_{ij}\lambda_i \leq c_j$, we have $\mathcal{L}_j(\boldsymbol{\lambda}) = 0$. In other words, the terms in the Lagrangian associated with column singletons can be ignored as long as we satisfy the dual constraint $a_{ij}\lambda_i \leq c_j$ for $i = \mathcal{I}(j)$.

With these insights, the equation dropping version of LP DASA is organized in the following way: Within the subiteration, we maintain two sets: B_k, the indices of bound columns, and D_k, the indices of dropped equations. In the line search where $\boldsymbol{v}_{k+1}$ is computed, we enforce the constraint $a_{ij}\lambda_i \leq c_j$, $i = \mathcal{I}(j)$, in the following way: If $a_{ij}\lambda_i$ reaches c_j at some point along the line segment $[\boldsymbol{v}_k, \boldsymbol{\omega}_k]$, then we add i to the dropped equation set, and we fix $\lambda_i = c_j/a_{ij}$. More precisely, in the line search of Algorithm 1, we replace the segment $[\boldsymbol{v}_k, \boldsymbol{\omega}_k]$ by a projected segment:

$$\mathbf{P}[\boldsymbol{v}_k, \boldsymbol{\omega}_k] = \{\mathbf{P}\boldsymbol{\omega} : \boldsymbol{\omega} \in [\boldsymbol{v}_k, \boldsymbol{\omega}_k]\},$$

where

$$(\mathbf{P}\boldsymbol{\lambda})_i = \begin{cases} c_j/a_{ij} & \text{if } a_{ij}\lambda_i \geq c_j \text{ and } i = \mathcal{I}(j), \\ \lambda_i & \text{otherwise.} \end{cases}$$

Once a row drops during the subiteration, it remains dropped for all the subsequent subiterations. At the start of the next iteration, before the k-loop, we initialize both B_0 and D_0. In this initialization phase, we check whether a fixed dual variable $\lambda_i = c_j/a_{ij}$, $i = \mathcal{I}(j)$, should be freed. We free λ_i if the dual function value increases after a small change in λ_i. This local behavior of the dual function is determined from the sign of the i-th component of the gradient (28). If $a_{ij} > 0$ and the i-th component of the gradient is < 0, then a small decrease in λ_i preserves dual feasibility ($a_{ij}\lambda_i \leq c_j$) and increases the value of the dual function. If $a_{ij} < 0$ and the i-th component of the gradient is > 0, then a small increase in λ_i preserves dual feasibility and increases the value of the dual function. Algorithm 3 provides a complete statement of the equation dropping version of LP DASA.

The proof (e.g. [20,23,25]) of finite convergence for Algorithm 1 also works for Algorithm 3. Finite convergence is due to the fact that certain sets cannot repeat. For Algorithm 3, the entity that cannot repeat is the pair (B_k, D_k) associated with the final dual subiteration.

The equation dropping strategy that we have developed can be implemented recursively. That is, when an equation is removed from $\mathbf{A}$, additional column singletons may be generated, leading to new equations that could drop. We did not implement this recursive procedure, but many of the test problems could be solved much faster using such an implementation. For example, in the Netlib test problems SCAGR7 and OSA, every single equation would drop in a recursive implementation. With AGG2, 505 out of 516 rows would drop, with SCTAP3, 1,365 out of 1,480 rows would drop, and with BNL2, 1,358 out of 2,265 rows would drop. Problems might also be presolved to expose additional column singletons. That is, a multiple of one equation could be subtracted from another equation

Algorithm 3 DASA with equation dropping

$l = 0$
$\boldsymbol{\lambda}_0$ = starting guess
while $\nabla \mathcal{L}(\boldsymbol{\lambda}_l) \neq \mathbf{0}$
  $\boldsymbol{v}_0 = \boldsymbol{\lambda}_l$
  $S = \{j \in [1,n] : (\boldsymbol{v}_0)_i = c_j/a_{ij}, i = \mathcal{I}(j)\}$
  $D_0 = \{i \in [1,m] : i = \mathcal{I}(j) \text{ for some } j \in S,\ a_{ij} \nabla \mathcal{L}(\boldsymbol{v}_0)_i \geq 0\}$
  $B_0 = \{j \notin S : x_j(\boldsymbol{\lambda}_l) = 0\} \cup \{j \in S : \mathcal{I}(j) \notin D_0\}$
  for $k = 0, 1, \ldots$
    $\boldsymbol{\omega}_k = \arg\max \{\mathcal{L}_{B_k^0}(\boldsymbol{\lambda}) : \boldsymbol{\lambda}_{D_k} = (\boldsymbol{v}_k)_{D_k}\}$
    $\boldsymbol{v}_{k+1} = \arg\max \{\mathcal{L}_{B_k^+}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \mathbf{P}[\boldsymbol{v}_k, \boldsymbol{\omega}_k]\}.$
    $B_{k+1} = \{j \in B_k : x_j(\boldsymbol{v}_{k+1}, B_k) = 0\}$
    $D_{k+1} = \{i : (\boldsymbol{v}_{k+1})_i = c_j/a_{ij},\ j \in \mathcal{I}^{-1}(i)\}$
    if $\boldsymbol{v}_{k+1} = \boldsymbol{\omega}_k$ break
  end
  $l = l + 1$
  $\boldsymbol{\lambda}_l = \boldsymbol{\omega}_k$
end

in order to create a column singleton. During the solution process, the row associated with the newly created column singleton could drop.

6 Numerical comparisons

In this section, we compare the performance of LP DASA to that of Simplex and Barrier methods as implemented in the commercial code CPLEX Version 9.1.3 [3], focusing on problems from the Netlib LP test set. Since the CPLEX Barrier code automatically applies a presolver to the input problem, we use the CPLEX presolved problem as input to our code. All the codes, both CPLEX and LP DASA, are written in C.

Both the first-order and second-order updates were implemented. Since the first-order update was generally more efficient than the second-order update, the numerical experiments, reported below, are based on the first-order update. Although Theorem 1 establishes convergence of the iteration (7), (8) for any choice of ϵ, it is observed numerically that for any fixed ϵ, convergence can be very slow; much faster convergence is achieved by decreasing ϵ after each iteration of (7), (8). The initial choice for ϵ and the factor used to reduce the size of ϵ after each proximal iteration were based on the number of rows m of $\mathbf{A}$:

$$\text{initial } \epsilon = \begin{cases} 2^{-6}, \text{ decay factor } 1/16, & \text{if } m < 100, \\ 2^{-3}, \text{ decay factor } 1/8, & \text{if } 100 \leq m < 2{,}500, \\ 1, \quad \text{decay factor } 1/4, & \text{otherwise.} \end{cases}$$

Qualitatively, we find that as the problem size increases, ϵ should increase. For any specific problem, ϵ can be fine-tuned to achieve faster convergence. For

example, the PDS test problems can be solved more quickly using a smaller initial ϵ than the choices given above.

Before starting the iteration (7), (8), we reorder the rows of $\mathbf{A}$ by applying a nested dissection ordering to minimize fill during a Cholesky factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$. We use CHOLMOD's nested dissection method [9,10,12,13], which uses METIS [28–30] to find node separators at each level in the nested dissection recursion, followed by a constrained column minimum degree ordering (a constrained version of COLAMD [7,8]).

If exact numerical cancellation is ignored, the fill-in associated with the matrix $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$ appearing in LP DASA is a subset of the fill-in associated with $\mathbf{A}\mathbf{A}^\mathsf{T}$. Hence, by minimizing the fill for $\mathbf{A}\mathbf{A}^\mathsf{T}$, we minimize the worst possible fill that could occur with $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$ for any choice of F.

To enhance numerical stability, we factorize the matrix $\mathbf{A}_F\mathbf{A}_F^\mathsf{T} + \sigma\mathbf{I}$, where $\sigma = 2^{-44}$ and the columns of $\mathbf{A}$ are scaled to be unit vectors. Since this value for σ is about 200 times the machine precision, the results of Saunders [38] suggest that solves should yield at least 2 correct digits. The factorization is implemented using routines from CHOLMOD. These routines use either a supernodal Cholesky factorization [34] or a row-oriented sparse Cholesky factorization algorithm [6,31], if the factorization is very sparse. The selection is made automatically; the supernodal factorization is used if the floating point operation count divided by the number of nonzeros in $\mathbf{L}$ is greater than 40. Our factorization routines are incorporated in MATLAB 7.2 as `chol`.

After a column is freed or a row is dropped, we use the sparse modification techniques developed in [9,10,12] to modify the factorization. The codes for modifying a factorization after column updates take advantage of speedup associated with multiple-rank updates. As a result, a rank 16 update of a sparse matrix achieves flop rates comparable to those of a BLAS dot product, and about 1/3 the speed of a BLAS matrix-matrix multiply [14], in the experiments given in [10]. Moreover, in the CHOLMOD versions of the update/downdate routine, we obtain further speedup by exploiting supernodes which are detected dynamically [13].

Our starting guesses for an optimal dual multiplier and primal solution are always $\boldsymbol{\lambda} = \mathbf{0}$ and $\mathbf{x} = \mathbf{0}$ respectively. Our convergence test is based on the LP optimality conditions. The primal approximation $\mathbf{x}(\boldsymbol{\lambda})$ in (29) always satisfies the bound constraints $\mathbf{x} \geq \mathbf{0}$. The column indices are expressed as $B \cup F$ where $F = B^c$. For all $j \in B$, $x_j(\boldsymbol{\lambda}) = 0$ and $(\mathbf{c} - \mathbf{A}^\mathsf{T}\boldsymbol{\lambda})_j \geq 0$. Hence, the optimality conditions would be satisfied if the primal and dual linear systems, $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{A}_F^\mathsf{T}\boldsymbol{\lambda} = \mathbf{c}_F$, were satisfied. Our convergence criterion is to stop when the relative residuals in the primal and dual systems satisfy the following condition:

$$\frac{\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_\infty}{1 + \|\mathbf{x}\|_\infty} + \frac{\|\mathbf{c}_F - \mathbf{A}_F^\mathsf{T}\boldsymbol{\lambda}\|_\infty}{1 + \|\boldsymbol{\lambda}\|_\infty} \leq 10^{-8} \tag{50}$$

Here $\|\cdot\|_\infty$ denotes the maximum absolute component of a vector. Since the test problems are given in MPS format, for which data oftenhas only 5 digit

Table 1 Numerical comparisons

LP DASA error			CPU time (s)		
Problem	**b**-error	**c**-error	LP DASA	Simplex	Barrier
PEROLD	3.0e-09	3.5e-13	1.16	0.15	0.12
25FV47	3.9e-09	8.0e-14	0.18	0.32	0.12
NUG07	2.7e-15	2.9e-18	0.30	0.29	0.12
PILOT_WE	4.1e-10	3.5e-10	1.53	0.54	0.16
PILOTNOV	1.6e-10	1.7e-13	1.21	0.23	0.18
CRE_C	1.3e-11	6.5e-12	0.44	0.18	0.20
CRE_A	8.0e-13	1.9e-15	0.61	0.18	0.25
OSA_07	5.4e-10	9.6e-17	0.22	0.23	0.43
NESM	1.7e-10	6.6e-09	0.39	0.35	0.19
D6CUBE	4.3e-11	1.0e-14	3.53	0.20	0.31
TRUSS	2.8e-10	9.0e-17	1.08	11.62	0.21
FIT2P	2.1e-18	3.7e-14	0.23	4.46	0.43
PILOT_JA	2.2e-09	7.5e-09	3.72	0.33	0.25
NUG08	3.8e-10	1.1e-14	0.71	1.86	0.25
FIT2D	1.4e-10	2.7e-15	0.26	0.59	0.71
GREENBEB	9.4e-09	4.3e-11	0.59	1.46	0.27
GREENBEA	4.6e-10	1.1e-09	1.82	1.03	0.28
80BAU3B	9.5e-09	9.0e-15	1.60	0.28	0.41
QAP8	1.9e-09	1.7e-13	0.84	1.40	0.28
KEN_11	4.9e-12	3.7e-16	0.53	0.38	0.55
DEGEN3	6.2e-10	3.2e-14	0.65	0.57	0.43
PDS_06	1.3e-11	4.3e-17	1.93	0.50	1.81
OSA_14	2.7e-14	7.3e-09	0.76	0.59	1.03
D2Q06C	7.7e-10	5.0e-09	2.70	2.81	0.62
MAROS_R7	2.1e-11	5.4e-12	0.83	3.29	1.37
STOCFOR3	5.6e-10	3.1e-09	5.24	1.75	1.09
PILOT	7.1e-14	9.6e-10	5.30	3.22	1.19
OSA_30	2.6e-14	6.6e-09	1.74	1.41	2.56
PDS_10	1.3e-11	3.8e-17	5.23	1.43	6.29
KEN_13	9.2e-11	1.0e-16	5.57	1.50	1.65
CRE_D	7.2e-09	5.4e-16	8.37	1.86	2.54
CRE_B	2.8e-11	6.4e-15	8.63	4.24	3.03
PILOT87	2.1e-13	4.1e-09	17.69	22.49	3.68
OSA_60	3.2e-13	6.9e-09	5.10	4.08	6.58
PDS_20	1.1e-11	2.1e-17	34.91	9.16	29.07
KEN_18	1.3e-16	3.9e-15	58.53	11.74	11.23
NUG12	5.0e-14	2.5e-09	52.86	202.72	11.33
QAP12	4.0e-09	5.1e-12	64.22	197.93	12.71
DFL001	5.3e-09	4.6e-09	36.94	22.51	16.04
NUG15	1.7e-15	7.1e-09	438.60	2787.11	78.99
QAP15	5.2e-09	1.4e-11	339.76	3143.31	85.83
NUG20	6.7e-12	7.0e-09	6094.90	–	13755.03
NUG30	7.8e-09	1.3e-09	32206.34	–	–

accuracy, our 8 digit error tolerance for the equation is, in a sense, much more accurate than most input data.

The test problems were solved on a 3.2 GHz Pentium 4 with 4GB of main memory. In Table 1 we compare the execution times in CPU seconds for all problems in the Netlib LP test set that required at least 0.1 s (based on the fastest CPLEX and LP DASA run times). The problems are ordered in accordance

Table 2 LP DASA statistics

Problem	Rows	Cols	Col+	Col−	Row+	Row−
PEROLD	503	1,273	6,410	2,037	219	678
25FV47	682	1,732	845	257	40	179
NUG07	473	930	378	240	4	6
PILOT_WE	602	2,526	12,126	1,323	197	1,071
PILOTNOV	748	1,999	4,124	2,180	336	560
CRE_C	2,257	5,293	1,889	209	96	1,284
CRE_A	2,684	6,382	1,181	160	135	1,098
OSA_07	1,047	24,062	298	249	95	11
NESM	598	2,764	2,023	363	104	638
D6CUBE	402	5,467	2,123	1,388	0	0
TRUSS	1,000	8,806	2,500	760	0	0
FIT2P	3,000	13,525	12	4	494	540
PILOT_JA	708	1,684	12,472	6,890	742	1,380
NUG08	741	1,631	396	352	6	8
FIT2D	25	10,388	940	811	9	17
GREENBEB	1,015	3,211	3,437	913	103	469
GREENBEA	1,015	3,220	12,699	1,731	332	2,194
80BAU3B	1,789	9,872	6,339	237	102	2,558
QAP8	741	1,631	340	256	16	26
KEN_11	5,511	11,984	1,750	1,053	2	5
DEGEN3	1,406	2,503	1,035	427	265	394
PDS_06	2,972	18,530	3,656	316	5	142
OSA_14	2,266	52,723	602	355	210	31
D2Q06C	1,855	5,380	5,731	601	125	1,616
MAROS_R7	2,152	6,578	25	50	67	74
STOCFOR3	8,388	15,254	3,725	666	833	5,736
PILOT	1,204	4,124	3,291	1,724	275	531
OSA_30	4,279	100,396	1,940	522	498	98
PDS_10	4,725	33,270	6,552	986	8	230
KEN_13	10,962	24,818	17,065	2,302	8	58
CRE_D	3,990	28,489	8,205	244	32	1,668
CRE_B	5,176	36,222	7,164	371	104	1,426
PILOT87	1,811	6,065	5,993	4,187	552	810
OSA_60	10,209	234,334	2,235	3,489	1,015	126
PDS_20	10,214	81,224	17,679	2,757	25	359
KEN_18	39,867	89,439	67,797	7,955	23	188
NUG12	2,793	8,855	2,411	1,749	192	291
QAP12	2,793	8,855	2,130	2,345	91	86
DFL001	3,861	9,607	2,216	1,722	36	55
NUG15	5,697	22,274	3,281	2,034	248	392
QAP15	5,697	22,274	4,459	2,993	97	146
NUG20	14,097	72,599	9,779	7,884	680	772
NUG30	52,260	379,350	–	–	–	–

with the fastest run times as reported in [11]. The first column gives the test problem name, the next two columns are the relative error terms in (50) associated with the LP DASA approximation. The final three columns give the execution time in seconds for the three codes. A dash means the problem cannot be solved by that method on our computer. The CPLEX Simplex method failed on the NUG20 since the solution time was essentially infinite. For both CPLEX Simplex and Barrier, there was not enough memory to solve NUG30. LP DASA, on the

Table 3 LP DASA solves and chols, Barrier chols, Simplex iterations

Problem	LP DASA solves	LP DASA chols	Barrier chols	Simplex iterations
PEROLD	3,019	71	44	1,120
25FV47	276	14	24	1,673
NUG07	186	6	13	2,071
PILOT_WE	4,349	154	46	2,546
PILOTNOV	1,734	28	20	1,250
CRE_C	541	41	28	2,053
CRE_A	415	46	28	2,255
OSA_07	69	6	18	523
NESM	705	28	38	2,266
D6CUBE	1,602	25	20	450
TRUSS	544	30	18	19,002
FIT2P	348	7	20	5,357
PILOT_JA	5,031	67	38	1,586
NUG08	149	6	9	6,417
FIT2D	124	8	18	159
GREENBEB	503	27	34	3,652
GREENBEA	2,299	72	38	3,591
80BAU3B	1,592	92	32	3,280
QAP8	145	8	10	5,522
KEN_11	261	13	20	6,647
DEGEN3	279	8	17	1,453
PDS_06	271	48	37	3,088
OSA_14	102	9	16	1,066
D2Q06C	1,349	38	30	4,160
MAROS_R7	73	1	10	2,797
STOCFOR3	1,074	25	31	6,569
PILOT	1,389	19	23	3,479
OSA_30	106	4	20	2,233
PDS_10	329	47	51	5,315
KEN_13	1,648	34	23	13,569
CRE_D	1,333	73	47	7,006
CRE_B	1,094	61	42	10,160
PILOT87	2,144	16	26	10,471
OSA_60	112	4	23	4740
PDS_20	646	65	41	18,025
KEN_18	4,576	53	29	49,173
NUG12	839	22	14	87,441
QAP12	660	15	17	81,099
DFL001	721	44	41	18,927
NUG15	610	26	15	430,970
QAP15	604	25	16	390,381
NUG20	3,849	16	23	–
NUG30	–	–	–	–

other hand, switches to an iterative implementation when there is insufficient memory to factor the matrix. In the iterative approach, developed in [24], (31) is solved using the SSOR preconditioned conjugate gradient scheme of Björck and Elfving [4].

At the following web site:

http://www.math.ufl.edu/∼hager/papers/LPDASA

we compare the objective function values obtained using the LP DASA code and a stopping criteria of the form (50), to the objective function values archived in Netlib: The 10^{-8} tolerance (50) often yields more than 8 digits agreement with the archived objective function values. The principal exceptions to this rule are the 7 pilot test problems (including `perold`) for which the agreement with the archived values ranged between 4 and 11 significant digits. Also, for the 16 test problems of Jeffrey Kennington, the objective function values are given to 8 significant digits, so for these test problems, we match the 8 given digits.

In Table 1, we see that for the 43 test cases, LP DASA was faster than the Barrier code (with crossover) for 11 problems, and was faster than the Simplex code (by default, dual Simplex) for 17 problems. LP DASA was fastest in problems where many equations drop. Our code was relatively slow for some of the PILOT problems, where there was significant growth in the error in the Cholesky factorization during downdates. Errors in the solution to the linear system increased the solution time since the search directions deviate from the true search directions.

Tables 2 and 3 give solution statistics for the three methods. The number of rows and columns given in Table 2 correspond to the presolved problems. In Table 2, we give the number of column updates (col+), column downdates (col−), row updates (row+), and row downdates (row−) employed by LP DASA. By a column update, we mean that we add a column to the current $\mathbf{A}_F$ and update the factorization. By a row update, we mean that we add a row to the current $\mathbf{A}_F$ and update the factorization. In Table 3 we give the number of times that LP DASA solved the system (31) or computed the Cholesky factorization of $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$. For comparison, the last two columns of Table 3 give the number of iterations of the CPLEX Barrier code, and the number of iterations of the Simplex code. Each iteration of the Barrier code requires the Cholesky factorization of a matrix with the sparsity pattern of $\mathbf{A}\mathbf{A}^\mathsf{T}$.

The data of Tables 2 and 3 show that LP DASA uses a comparable number of factorizations to Barrier; for 20 problems, LP DASA used fewer factorizations, while in 23 problems, Barrier used fewer factorizations. Each factorization in LP DASA requires less work than a Barrier factorization since LP DASA factorizes $\mathbf{A}_F\mathbf{A}_F^\mathsf{T}$, which can be much sparser than the $\mathbf{A}\mathbf{D}^2\mathbf{A}^\mathsf{T}$ matrix ($\mathbf{D}$ is a diagonal matrix) that CPLEX Barrier must factor. In LP DASA, the matrix is factored when the rank of the update is relatively large. Hence, there is a trade-off between factorizations and updates/downdates – by increasing the number of factorizations, we can reduce the number of updates and downdates. Generally, LP DASA uses much fewer solves than the Simplex code; also, the number of solves in LP DASA is often much less than the number of updates and downdates – in each iteration, the line search can free many variables.

In Fig. 1 we study the convergence towards optimality of the three methods using Jeffrey Kennington's test problem `pds_10`. The horizontal axis is the relative CPU time (CPU time divided by the total solution time), while the vertical axis is $\log_{10}$ of the relative error in the cost. For LP DASA, the relative error is computed for the primal approximation associated with the dual iterate in (7). The convergence of LP DASA is initially similar to that of the Barrier

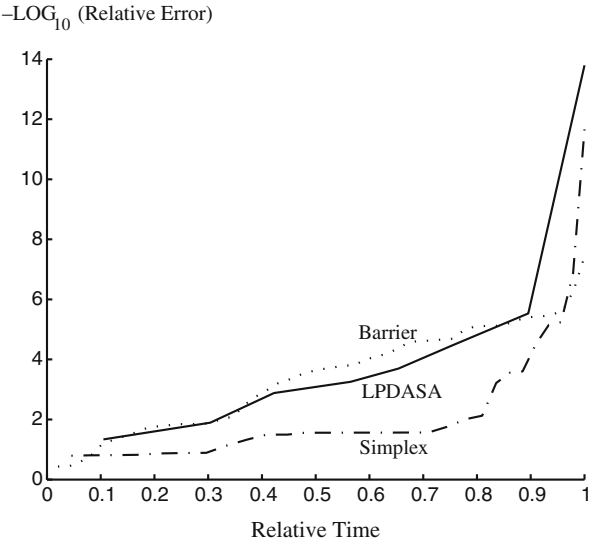

Fig. 1 Accuracy of solution versus relative CPU time for PDS_10

method. On the other hand, with LP DASA, there are no conditioning problems in a neighborhood of the solution and a crossover routine is not needed. Full accuracy, on the order of the machine epsilon, can be computed while barrier methods typically are limited to accuracy on the order of the square root of the machine epsilon. With the Simplex method, 2-digit accuracy is not achieved until the computer run is more than 75% complete. With both Barrier and LP DASA, 2-digit accuracy is achieved when the run is 30% complete. In a similar study in [24], we saw that when iterative methods are used to solve (31), we obtain 2-digit accuracy in a small fraction of the total computing time.

In summary, LP DASA is able to compete with today's state-of-the-art commercial software. The same LP DASA platform can be run using either an iterative solver or a factorization-based solver to obtain solutions of high relative accuracy. Equation dropping techniques significantly lower the overall solution times. Recursive implementations of the equation dropping techniques should further reduce solution times. Another approach for further reducing solution times is to implement the algorithm in a multilevel fashion. This multilevel approach is developed in [11]. The solution times reported in the present paper were obtained by running the multilevel code using a single level.

Acknowledgments Initial work connected with the application of DASA to linear programming appears in the thesis of Chun-Liang Shih [39]. Experimentation with a very early version of the code was done by Erik Lundin, a student participating in an exchange program between the Center for Applied Optimization at the University of Florida and the Royal Institute of Technology (KTH) in Sweden. Constructive comments by the referees led to a much improved presentation. In particular, the elegant reformulation of the LP version of Algorithm 2 in terms of least squares problems was suggested by a referee.

References

1. Andersen, E.D., Andersen, K.D.: Presolving in linear programming. Math. Program. **71**, 221–245 (1995)
2. Bergounioux, M., Kunisch, K.: Primal-dual strategy for state-constrained optimal control problem. Comput. Optim. Appl. **22**, pp. 193–224 (2002)
3. Bixby, R.E.: Progress in linear programming. ORSA J. Comput. **6**, 15–22 (1994)
4. Björck, A., Elfving, T.: Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. BIT **19**, 145–163 (1979)
5. Clarke, F.H.:Generalized gradients and applications. Trans. Am. Math. Soc. **205**, 247–262 (1975)
6. Davis, T.A.: Algorithm 849: a concise sparse Cholesky factorization package. ACM Trans. Math. Softw. **31**, 587–591 (2005)
7. Davis, T.A., Gilbert, J.R., Larimore, S.I., Ng, E.G.: Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**, 377–380 (2004)
8. Davis, T.A., Gilbert, J.R., Larimore, S.I., Ng, E.G.: A column approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**, 353–376 (2004)
9. Davis, T.A., Hager, W.W.: Modifying a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **20**, 606–627 (1999)
10. Davis, T.A., Hager, W.W.: Multiple-rank modifications of a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **22**, 997–1013 (2001)
11. Davis, T.A., Hager, W.W.: Dual multilevel optimization, to appear in Mathematical Programming University of Florida (2004)
12. Davis, T.A., Hager, W.W.: Row modifications of a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **26**, 621–639 (2005)
13. Davis, T.A., Hager, W.W., Chen, Y.C., Rajamanickam, S.: CHOLMOD: a sparse Cholesky factorization and modification package, ACM Trans. Math. Softw. (2006) (in preparation)
14. Dongarra, J.J., Du Croz, J.J., Duff, I.S., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**, 1–17 (1990)
15. Ekeland, I., Temam, R.: Convex Analysis and Variational Problems. North-Holland, Amsterdam (1976)
16. Gill, P.E., Saunders, M.A., Shinnerl, J.R.: On the stability of cholesky factorization for quasi-definite systems. SIAM J. Matrix Anal. Appl. **17**, 35–46 (1996)
17. Golub, G.H., Loan, C.F.V.: Unsymmetric positive definite linear systems. Linear Algebra Appl. **28**, 85–98 (1979)
18. Hager, W.W.: Convex control and dual approximations, part I. Control Cybern. **8**, 5–22 (1979)
19. Hager, W.W.: Inequalities and approximation. In: Coffman, C.V., Fix, G.J. (eds.) Constructive Approaches to Mathematical Models pp. 189–202. (1979)
20. Hager, W.W.: The dual active set algorithm. In: Pardalos, P.M. (ed.) Advances in Optimization and Parallel Computing pp. 137–142. North Holland, Amsterdam (1992)
21. Hager, W.W.: The LP dual active set algorithm. In: Leone, R.D., Murli, A., Pardalos, P.M., Toraldo, G.: (eds.) High Performance Algorithms and Software in Nonlinear Optimization pp. 243–254. Kluwer, Dordrecht, (1998)
22. Hager, W.W.: Iterative methods for nearly singular linear systems. SIAM J. Sci. Comput. **22**, 747–766 (2000)
23. Hager, W.W.: The dual active set algorithm and its application to linear programming. Comput. Optim. Appl. **21**, 263–275 (2002)
24. Hager, W.W.: The dual active set algorithm and the iterative solution of linear programs. In: Pardalos, P.M., Wolkowicz, H. (eds.) Novel Approaches to Hard Discrete Optimization, vol. 37, pp. 95–107 Fields Institute Communications, (2003)
25. Hager, W.W., Hearn, D.W.: Application of the dual active set algorithm to quadratic network optimization. Comput. Optim. Appl. **1**, 349–373 (1993)
26. Hager, W.W., Ianculescu, G.: Dual approximations in optimal control. SIAM J. Control Optim. **22**, 423–465 (1984)
27. Hager, W.W., Shi, C.-L., Lundin, E.O.: Active set strategies in the LP dual active set algorithm, tech. report. University of Florida, http://www.math.ufl.edu/~hager/LPDASA (1996)
28. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**, 359–392 (1998)

29. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**, 96–129 (1999)
30. Karypis, G., Kumar, V.: Parallel multilevel k-way partitioning scheme for irregular graphs. SIAM Rev. **41**, 278–300 (1999)
31. Liu, J.W.H.: A generalized envelope method for sparse factorization by rows. ACM Trans. Math. Softw. **17**, 112–129 (1991)
32. Martinet, B.: Régularisation d'inéquations variationnelles par approximations successives. Rev. Francaise Inform. Rech. Oper. Ser. R-3, **4**, 154–158 (1970)
33. Martinet, B.: Determination approachée d'un point fixe d'une application pseudo-contractante. Comptes Rendus des Séances de l'Académie des Sciences, **274**, 163–165 (1972)
34. Ng, E.G., Peyton, B.W.: Block sparse Cholesky algorithms on advanced uniprocessor computers. SIAM J. Sci. Comput. **14**, 1034–1056 (1993)
35. Pan, P.Q.: A dual projective pivot algorithm for linear programming. Comput. Optim. Appl. **29**, 333–346 (2004)
36. Rockafellar, R.T.: Monotone operators and the proximal point algorithm. SIAM J. Control, **14**, 877–898 (1976)
37. Roos, C., Terlaky, T., Vial, J.-P.: Theory and Algorithms for Linear Optimization: An Interior Point Approach. Wiley, New York (1997)
38. Saunders, M.A.: Cholesky-based methods for sparse least squares: The benefits of regularization. In: Adams, L., Nazareth, J.L. (eds.) Linear and Nonlinear Conjugate Gradient-Related Methods, pp. 92–100. SIAM, Philadelpha (1996)
39. Shih, C.L.: Active Set Strategies in Optimization. PhD Thesis, University of Florida, Department of Mathematics (1995)
40. Vanderbei, R.J.: Symmetric quasi-definite matrices. SIAM J. Optim. **5**, 100–113 (1995)
41. Volkwein, S.: Lagrange-SQP techniques for the control constrained optimal boundary control for Burger's equation. Comput. Optim. Appl. **26**, 253–284 (2003)