Computational Optimization and Applications, 1, (1993), 349–373
© 1993 Kluwer Academic Publishers. Manufactured in The Netherlands.

Application of the Dual Active Set Algorithm to Quadratic Network Optimization

WILLIAM W. HAGER
Department of Mathematics, University of Florida, Gainesville, FL 32611

DONALD W. HEARN
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611

Received May 6, 1991; Revised August 14, 1992

Abstract. A new algorithm, the dual active set algorithm, is presented for solving a minimization problem with equality constraints and bounds on the variables. The algorithm identifies the active bound constraints by maximizing an unconstrained dual function in a finite number of iterations. Convergence of the method is established, and it is applied to convex quadratic programming. In its implementable form, the algorithm is combined with the proximal point method. A computational study of large-scale quadratic network problems compares the algorithm to a coordinate ascent method and to conjugate gradient methods for the dual problem. This study shows that combining the new algorithm with the nonlinear conjugate gradient method is particularly effective on difficult network problems from the literature.

Keywords: dual algorithm, active set algorithm, conjugate gradients, quadratic networks, proximal point method

1. Introduction

We consider the problem

$$\text{minimize } f(x) \text{ subject to } h(x) = 0,\ l \le x \le u, \tag{1}$$

where f and h are continuously differentiable with f real-valued and h mapping $\mathbb{R}^n$ to $\mathbb{R}^m$. The n-vectors l and u are constant upper and lower bounds on the vector x. Hence, the constraints in (1) consist of both equality constraints

$$h(x) = 0, \tag{2}$$

and inequality bound constraints

$$l \le x \le u. \tag{3}$$

(Components of l or u can take the values $-\infty$ or $+\infty$ respectively.) Introducing a multiplier $\lambda \in \mathbb{R}^m$ for the equality constraint in (2), we obtain the Lagrangian L defined by

$$L(\lambda, x) = f(x) + \lambda^T h(x), \tag{4}$$

and the associated dual functional

$$L(\lambda) = \inf L(\lambda, x) \text{ subject to } l \leq x \leq u. \tag{5}$$

A fundamental new algorithm, the dual active set algorithm, is presented for solving the *unconstrained* dual problem

$$\text{maximize } L(\lambda) \text{ subject to } \lambda \in \mathbb{R}^m. \tag{6}$$

The initial algorithm that we present is conceptual in the sense that the various minimizers and maximizers appearing in an iteration may not always exist. The implementable version of the algorithm is embedded in the proximal point method.

The dual active set algorithm is motivated by the fact that for each $\lambda \in \mathbb{R}^m$, the evaluation of $L(\lambda)$ in (5) yields a minimizing $x = x(\lambda)$, which identifies an *active set* of bound constraints (3), that is, those constraints for which $x_i(\lambda)$ is equal to either l_i or u_i. Starting from an arbitrary λ_0, the dual active set algorithm obtains a solution λ^* to the unconstrained optimization problem (6) in a finite number of iterations. Under appropriate hypotheses, there exists a minimizing $x = x(\lambda^*)$ in (5), corresponding to $\lambda = \lambda^*$, which is a solution to the primal problem (1). Hence, in a finite number of iterations, the algorithm identifies the active set of bound constraints associated with an optimal solution. This algorithm is similar in spirit to the well-known primal active set method ([35]) that determines a solution of (1), and an optimal active set, by evaluating a finite number of primal iterates that satisfy the constraints (2) and (3). However, there are major differences in these two distinct approaches that will be examined later in the context of quadratic programming.

The remainder of this paper is organized as follows. In Section 2, we present the conceptual form of the dual active set algorithm, which is based on the work of Hager [24] and [25]. Finite convergence of the method is established, and the differences between dual and primal active set methods are discussed. Section 3 gives the implementable form of the algorithm. Section 4 specializes the method to convex quadratic programming, including quadratic networks. Section 5 discusses how conjugate gradient techniques are applied to the quadratic network dual. Finally, Section 6 gives the results of our numerical experiments with quadratic network problems from the literature.

2. The dual active set algorithm

To emphasize the generality of the dual active set algorithm, we first present it in its conceptual form. That is, it will be assumed that all entities introduced in the statement of the algorithm exist, and can be readily calculated. As mentioned above, implementation techniques are dealt with in the following sections.

To solve the dual problem (6), the dual active set algorithm employs two auxiliary functions that depend on a subset B of the indices $\{1, 2, \cdots, n\}$. Given a vector x, let x_B denote the vector with components x_i associated with indices $i \in B$. Given a vector $z \in \mathbb{R}^n$, we define:

$$L_B(\lambda) = \inf L(\lambda, x) \text{ subject to } l_B \leq x_B \leq u_B, \tag{7}$$

and

$$L_B^z(\lambda) = \inf L(\lambda, x) \text{ subject to } x_B = z_B. \tag{8}$$

Note that the variable x_i in (7) and (8) is unconstrained if $i \notin B$. $L_B^z(\lambda)$ is defined by fixing x_B and carrying out the Lagrangian minimization in (8) with respect to the "free" variables x_i, $i \notin B$. As we will see in Section 4, $L_B^z(\lambda)$ is often much smoother than $L(\lambda)$, and the maximum of $L_B^z(\lambda)$ over λ is much easier to evaluate than the maximum of $L(\lambda)$. Within an iteration of the dual active set algorithm, we adjust the set B until the maxima of $L_B^z(\lambda)$ and $L_B(\lambda)$ coincide. From the inequality $L_B(\lambda) \leq L(\lambda)$ for any B and all λ, it follows that the maximum of $L_B(\lambda)$ is a lower bound for the maximum of $L(\lambda)$. In successive iterations of the dual active set algorithm, the set B and the vector z are adjusted until the maxima of L_B and L are also equal.

The strategy of the iteration is as follows: At any dual iterate λ_k, z is given by

$$z = x(\lambda_k) = \arg\min L(\lambda_k, x) \text{ subject to } l \leq x \leq u,$$

while the set B is

$$B = \{i : z_i = l_i \text{ or } z_i = u_i\}.$$

The maximizer of L_B^z is denoted by μ:

$$\mu = \arg\max L_B^z(\lambda) \text{ subject to } \lambda \in \mathbb{R}^m.$$

Since L_B^z is concave, $L_B^z(\lambda) \geq L_B^z(\lambda_k)$ whenever λ lies on the line segment connecting λ_k and μ. At $\lambda = \lambda_k$, $L_B^z(\lambda)$ and $L_B(\lambda)$ are equal. Starting at λ_k, the direction $\mu - \lambda_k$ is searched until the first point is found where L_B^z and L_B are no longer equal. At this point, B is updated and the process continues through a sequence of subiterates to produce λ_{k+1}. In the precise statement of the iteration that follows, subiterates between λ_k and λ_{k+1} are denoted by ν_j.

Dual active set iteration:

Given λ_k, let $j = 0$, $\nu_0 = \lambda_k$, and define

$$B_0 = \{i : z_i = l_i \text{ or } z_i = u_i\}$$

where

$$z = x(\lambda_k) = \arg\min L(\lambda_k, x) \text{ subject to } l \leq x \leq u.$$

Subiteration: Let μ_j maximize $L^{z}_{B_j}(\lambda)$ over λ and define $\mu(t) = \nu_j + t(\mu_j - \nu_j)$.

Determine the largest interval $[0, \bar{t}]$, $\bar{t} \geq 0$, such that

$$L^{z}_{B_j}(\mu(t)) = L_{B_j}(\mu(t)) \text{ for every } t \in [0, \bar{t}]. \tag{9}$$

If $\bar{t} < 1$, then put $\nu_{j+1} = \mu(\bar{t})$. The set B_{j+1} is obtained by deleting from B_j those indices $i \in B_j$ with the property that

$$\left.\frac{\partial L(\lambda, x)}{\partial x_i}\right|_{\substack{\lambda=\bar{\lambda}\\ x=\bar{x}}} = 0,$$

where $\bar{x}$ is a minimizer in (8) associated with $\bar{\lambda} = \mu(\bar{t})$. Increment j and repeat the subiteration.

If $\bar{t} \geq 1$, then set $\lambda_{k+1} = \mu_j$, increment k, and proceed to the next iteration.

Since the dual problem is unconstrained, the dual active set iterations terminate when $\nabla L(\lambda_k) = 0$ for some λ_k. By the concavity of the dual functional, $\lambda_k = \lambda^*$ at termination. The convergence of this scheme is examined under a strong convexity assumption. That is, we assume that there exists a constant $\alpha > 0$ such that

$$L(\lambda, y) \geq L(\lambda, x) + \nabla_x L(\lambda, x)(y - x) + \alpha\|y - x\|^2, \tag{10}$$

where α is independent of x, y, and λ; and where $\|.\|$ denotes the Euclidean norm.

THEOREM 1. *If f and h are continuously differentiable on $I\!R^n$, L satisfies the strong convexity assumption (10), and there exists a maximizer μ_j of $L^{z}_{B_j}$ for each j, then the dual active set algorithm reaches a solution of (6) in a finite number of iterations and subiterations.*

Proof. The proof has the following structure: We show that B_{j+1} is strictly contained in B_j so that the subiteration eventually terminates. Then we show that the final B_j set generated in the subiteration does not repeat so that the algorithm reaches a solution to the dual problem in a finite number of iterations.

Finite termination of the subiteration

By the convexity assumption, $L^{z}_{B_0}(\lambda_k) = L_{B_0}(\lambda_k) = L(\lambda_k)$. By the continuous differentiability of f and h and by (10), there exists a minimizer $x(t)$ in (7) associated with $\lambda = \mu(t)$ and $B = B_j$, and $x(t)$ is a continuous function of t (see [20, Theorem 4.1]). Since $L^{z}_{B_j}(\mu(t)) = L_{B_j}(\mu(t))$ for $0 \leq t \leq \bar{t}$, and since

(8) also has a unique minimizer, which is feasible in (7), we conclude that $x(t)$ is the minimizer in (8) for $0 \le t \le \bar{t}$. It follows that $x(\bar{t})_{B_j} = \bar{x}_{B_j} = z_{B_j}$. Since $L^z_{B_j}(\mu(t)) > L_{B_j}(\mu(t))$ as t extends beyond $\bar{t}$, we conclude that $x(t)$ becomes infeasible for (8) as t extends beyond $\bar{t}$. Since $x(t)$ is feasible for (7), we see that $l_i < x_i(t) < u_i$ for some $i \in B_j$ and for $t > \bar{t}$ with t near $\bar{t}$. Since both the constraints $x_i \ge l_i$ and $x_i \le u_i$ are inactive at $x = x_i(t)$, the first-order necessary conditions imply that

$$\left.\frac{\partial L(\lambda, x)}{\partial x_i}\right|_{\substack{\lambda = \mu(t) \\ x = x(t)}} = 0$$

for $t > \bar{t}$ with t near $\bar{t}$. Since $x(t)$ and $\mu(t)$ are continuous functions of t, we let t approach $\bar{t}$ to obtain

$$\left.\frac{\partial L(\lambda, x)}{\partial x_i}\right|_{\substack{\lambda = \bar{\lambda} \\ x = \bar{x}}} = 0, \tag{11}$$

where $\bar{x} = x(\bar{t})$. Hence, B_{j+1} is strictly contained in B_j, and the subiterations terminate in a finite number of steps.

Convergence to optimality

Since $L^z_B(\lambda)$ is a concave function of λ for any choice of B and μ_j maximizes $L^z_{B_j}(\lambda)$ over λ, we have

$$L^z_{B_j}(\nu_j) \le L^z_{B_j}(\nu_{j+1}).$$

By the convexity assumption (10), the first-order necessary conditions associated with (8) are sufficient for optimality. Since $\bar{x}$ satisfies the first-order necessary conditions associated with (8) and with the choice $B = B_j$ and $\lambda = \bar{\lambda}$, and since (11) holds for each $i \in B_j \backslash B_{j+1}$, $\bar{x}$ also satisfies the first-order necessary conditions associated with (8) and the choice $B = B_{j+1}$ and $\lambda = \bar{\lambda}$. It follows that

$$L^z_{B_j}(\nu_{j+1}) = L^z_{B_{j+1}}(\nu_{j+1}).$$

If $\overline{B}$ denotes the final set B_j generated by the subiterations, then by (9) we have

$$L^z_{\overline{B}}(\lambda_{k+1}) = L_{\overline{B}}(\lambda_{k+1}).$$

Since $L_B(\lambda) \le L(\lambda)$ for any choice of B, we conclude that

$$L^z_{B_{j+1}}(\nu_{j+1}) \le L(\lambda_{k+1}).$$

Combining these inequalities gives

$$L(\lambda_k) \le L^z_{B_0}(\nu_1) \le L^z_{B_1}(\nu_2) \le \cdots \le L^z_{\overline{B}}(\lambda_{k+1}) = L_{\overline{B}}(\lambda_{k+1}) \le L(\lambda_{k+1}). \quad (12)$$

If the algorithm has not terminated at iteration k, then $\nabla L(\lambda_k) \neq 0$, and thus

$$\nabla L(\lambda_k) = h(z) = \nabla L_{B_0}(\lambda_k) = \nabla L_{B_0}(\nu_0) \neq 0.$$

(See Clarke [7, Proposition 1.13 and Theorem 2.1.]) Since $L^z_{B_0}(\lambda)$ is a concave function of λ, μ_0 maximizes $L^z_{B_0}$, and ν_1 lies between ν_0 and μ_0, it follows that $L^z_{B_0}(\nu_0) < L^z_{B_0}(\nu_1)$ unless $\nu_0 = \nu_1$. However, in this special case,

$$\left.\frac{\partial L(\lambda, x)}{\partial x_i}\right|_{\substack{\lambda = \lambda_k \\ x = z}} \neq 0$$

for each $i \in B_1$, which implies that $L^z_{B_1}(\nu_1) < L^z_{B_1}(\nu_2)$. Hence, by (12), we have

$$L(\lambda_k) < L^z_{\overline{B}}(\lambda_{k+1}) \le L(\lambda_{k+1}).$$

Since the components of $z_{\overline{B}}$ are chosen from a finite set, the dual active set algorithm reaches a maximizer of (6) in a finite number of iterations. □

Studying the convergence proof, we make two observations:

1. In the final subiteration, λ_{k+1} can be chosen to maximize L along the line through ν_j and μ_j – all the inequalities in the proof remain valid.
2. The subiterations can be terminated whenever a point is found for which the value of L increases. This leads to the following quick step form of the iteration.

Dual active set iteration, quick step:

The iteration is identical to the previous iteration; however, the subiteration terminates whenever $L(\mu_j) > L(\lambda_k)$, and we set $\lambda_{k+1} = \mu_j$ (or alternatively, we maximize L along the line through ν_j and μ_j to get λ_{k+1}).

It is interesting to contrast the dual active set algorithm with the primal active set algorithm which, as noted by Luenberger [35, p. 425] is often employed for quadratic programs where $f(x)$ is quadratic and $h(x) = Ax - b$ is linear. Since the latter method is described in detail in this reference, we only highlight its attributes:

- Each iterate x_k satisfies all constraints.
- Each iteration attempts to minimize f over a working set of constraints defined by $Ax = b$, and a subset B of the bound constraints treated as equalities.

- If, in the process of optimizing over the current working set, some x_i for $i \notin B$ encounters its lower or upper bound, the working set is expanded and the optimization continues over the updated working set.
- Once the optimal solution is found for a given working set, the multipliers associated with the bound constraints are computed, and if all have the appropriate sign to satisfy the Karush-Kuhn-Tucker conditions, the algorithm terminates. Otherwise, bound constraints associated with multipliers of incorrect sign are dropped from the working set, and the algorithm continues.

Of course, it is possible to introduce the multipliers for the bound constraints as well as the λ multipliers for the constraint $Ax = b$, construct a *constrained* dual problem, and apply the primal active set algorithm to this dual problem. (See Goldfarb and Idnani [18, p. 24] for an interesting discussion of algorithmic strategies applied to primal and dual formulations of quadratic programs.)

The dual active set algorithm, by contrast, maximizes the objective $L(\lambda)$ of an unconstrained dual problem. It has the following features:

- The bound constraints are maintained at each iteration, while the constraint $Ax = b$ is relaxed, until termination.
- It is the dual iterates λ_k that force the selection of the set B and that identify those $x_i(\lambda_k)$ at a bound. The multipliers of the bound constraints are not used in the computation.
- The move from iterate λ_k to λ_{k+1} is more involved than a line search step; in particular, the algorithm utilizes certain approximations to $L(\lambda)$ which change during the move as the set B changes.
- In an implementable form of the algorithm, developed in the next section, the approximations to $L(\lambda)$ include the use of the proximal point method, without which the iterations could diverge.

Despite these differences, the dual active set algorithm shares one important feature of the primal active set algorithm: It produces the primal active set in a finite number of iterations as proven in Theorem 1.

3. The dual active set algorithm: Implementable form

The existence of the maximizer μ_j in the conceptual form of the dual active set algorithm is related to the indices in B_j. In particular, whenever the set

$$\{x \in I\!R^n : h(x) = 0,\ x_{B_j} = z_{B_j}\}$$

is empty, μ_j may not exist. For the large network optimization problems solved in Section 6, we observed that in every test problem, a set B_j was encountered for which the maximizer μ_j failed to exist. In this section, we "regularize" L by the addition of a strongly concave term to ensure the existence of a maximum.

Letting ε denote a positive regularization parameter and letting Λ denote a fixed vector in $\mathbb{R}^m$, one of the simplest regularized functions is

$$M(\lambda) = L(\lambda) - \varepsilon\|\lambda - \Lambda\|^2.$$

The regularized forms of the modified functions are

$$M_B(\lambda) = L_B(\lambda) - \varepsilon\|\lambda - \Lambda\|^2 \text{ and } M_B^z(\lambda) = L_B^z(\lambda) - \varepsilon\|\lambda - \Lambda\|^2.$$

In the regularized problem, the maximization of L is replaced by a maximization of M:

$$\text{maximize } M(\lambda) \text{ subject to } \lambda \in \mathbb{R}^m. \tag{13}$$

Maximizing the regularized function M rather than L is the essence of the proximal point algorithm. This scheme has been studied extensively in the literature; some references include the papers [37] and [38] by Martinet, [45] and [46] by Rockafellar, [36] by Luque, [47] by Spingarn, and [19] by Ha. If Λ_k is the current iterate in the proximal point algorithm, then the new iterate Λ_{k+1} is computed by maximizing the M associated with the choice $\varepsilon = \varepsilon_k$ and $\Lambda = \Lambda_k$. If the ε_k are sufficiently small, then under suitable assumptions, the proximal point algorithm converges linearly; if the ε_k tend to zero, then the convergence is superlinear. For example, if there exists a neighborhood Ω of λ^* and an $\alpha > 0$ such that

$$L(\lambda) \leq L(\lambda^*) - \alpha\|\lambda - \lambda^*\|^2 \tag{14}$$

for every $\lambda \in \Omega$, then we have (see [46]):

LEMMA 1. *If (14) holds and the ε_k are uniformly bounded, then for k sufficiently large, Λ_{k+1} satisfies the inequality*

$$\|\Lambda_{k+1} - \lambda^*\| \leq \frac{\varepsilon_k}{\varepsilon_k + \alpha}\|\Lambda_k - \lambda^*\|.$$

Hence, the Λ_k approach λ^* superlinearly if ε_k tends to zero.

The dual active set algorithm can be used to maximize M. Again, if λ_k is the current iterate and z denotes an x in (5) that attains the minimum when $\lambda = \lambda_k$, then λ_{k+1} is computed through subiterations that start with the initialization $\nu_0 = \lambda_k$ and B_0 = the set of active indices of z. The iteration becomes the following.

Dual active set iteration (regularized form):

The iteration is identical to the previous one except that μ_j is chosen to maximize $M_{\bar{B}_j}^z(\lambda)$ over λ, and $M_{B_j}(\lambda)$ replaces $L_{B_j}(\lambda)$. If a quick step is employed, then the subiteration is terminated whenever $M(\mu_j) > M(\lambda_k)$, and we set $\lambda_{k+1} = \mu_j$ (or alternatively, we maximize M along the line through ν_j and μ_j to get λ_{k+1}).

Since $M_{B_j}^z$ is strongly concave, it has a maximum. The same proof given for the original dual active set algorithm, also applies to the regularized form, although each L in the former proof must be replaced by M. Hence, we state

THEOREM 2. *If f and h are continuously differentiable on $I\!R^n$ and L satisfies the strong convexity assumption (10), then the regularized version of the dual active set algorithm reaches a solution of (13) in a finite number of iterations and subiterations.*

4. Quadratic programs and quadratic networks

Now we examine the dual active set algorithm in the context of quadratic programming, examining in detail various implementation issues and the specialization to quadratic networks. We consider the optimization problem (1) in the case where

$$f(x) = \frac{1}{2}x^T Qx + c^T x \text{ and } h(x) = Ax - b; \tag{15}$$

here A is an $m \times n$ matrix; and Q is a symmetric, positive definite $n \times n$ matrix. Observe that a quadratic program of the form

$$\text{minimize } \frac{1}{2}x^T Qx + c^T x \text{ subject to } Cx \leq d \tag{16}$$

is equivalent to the program

$$\text{minimize } \frac{1}{2}x^T Qx + \frac{1}{2}\|Cx + y - d\|^2 + c^T x \text{ subject to } Cx + y = d,\ y \geq 0.$$

Hence, if Q is positive definite, then (16) is equivalent to a quadratic program in the form (15) with positive definite Hessian.

Now let us obtain a formula for the dual function L_B^z. Given a subset B of the indices $\{1, 2, \cdots, n\}$, let N denote the complement of B. After partitioning Q, A, and c in the natural way, L_B^z is expressed

$$\begin{aligned} L_B^z(\lambda) &= \frac{1}{2}z_B^T Q_{BB} z_B + c_B^T z_B + \lambda^T(A_B z_B - b) \\ &+ \inf_{x_N} \frac{1}{2}x_N^T Q_{NN} x_N + x_N^T Q_{NB} z_B + c_N^T x_N + \lambda^T A_N x_N. \end{aligned} \tag{17}$$

Here Q_{ST} denotes the submatrix of Q formed by the intersection of rows associated with indices $i \in S$ and columns associated with indices $j \in T$. The matrix A_N is the submatrix of A formed by the columns associated with indices $i \in N$. And after carrying out the minimization, we have

$$\begin{aligned} L_B^z(\lambda) &= \frac{1}{2}z_B^T Q_{BB} z_B + c_B^T z_B + \lambda^T(A_B z_B - b) \\ &- \frac{1}{2}(A_N^T\lambda + c_N + Q_{NB}z_B)^T Q_{NN}^{-1}(A_N^T\lambda + c_N + Q_{NB}z_B). \end{aligned}$$

Computation of μ

Computing the point μ in the dual subiteration is equivalent to solving a linear system of equations whose coefficient matrix is $A_N Q_{NN}^{-1} A_N^T$. In successive subiterations, we add a column to A_N, and we add both a row and a column to Q_{NN}. An important numerical issue is how to evaluate μ efficiently after adding these rows and column. As we now show, these additions yield a rank-one change in $A_N Q_{NN}^{-1} A_N^T$. Hence, its inverse can be updated using Sherman-Morrison-Woodbury techniques—see [23] for a discussion of these techniques.

We use $^+$ superscript to denote the matrix generated after adding a column to A_N and a row and a column to Q_{NN} (the new rows and column are placed on the borders of the matrices). Suppose that Q is positive definite and let C_{NN} denote the lower triangular Cholesky factor of Q_{NN}. Then C_{NN}^+ is the same as C_{NN} except for the addition of a new row along the bottom and zeros along the right border. Likewise, the inverse of C_{NN}^+ is the same as the inverse of C_{NN} except for the addition of a new row along the bottom and zeros along the right border. If v^T denotes the last row of the inverse of C_{NN}^+, then

$$(A_N Q_{NN}^{-1} A_N^T)^+ = A_N Q_{NN}^{-1} A_N^T + ww^T \text{ where } w = A_N^+ v.$$

Thus, in each subiteration of the dual active set algorithm, the coefficient matrix of the linear system associated with the computation of μ is modified by a rank-one term.

Computation of $\bar{t}$

The parameter $\bar{t}$ in (9) can be evaluated in the following way. Let $\mu(t)$ denote the vector appearing in the statement of the dual active set iteration, and let $x(t)$ be defined as follows:

$$x_i(t) = z_i \text{ for } i \in B \text{ and } x_i(t) = -\left[Q_{NN}^{-1}\left(Q_{NB}z_B + c_N + A_N^T \mu(t)\right)\right]_i \text{ for } i \in N.$$

Observe that $x(t)$ is the vector that attains the minimum in (8) when $\lambda = \mu(t)$. Since $\mu(t)$ is a linear function of t, $x(t)$ is also a linear function of t. For each value of i, let t_i denote the root of the linear equation

$$\left(Qx(t) + c + A^T \mu(t)\right)_i = 0, \tag{18}$$

when a root exists. When the coefficient of t in (18) vanishes, set $t_i = \infty$. With these definitions, $\bar{t}$ is expressed:

$$\bar{t} = \underset{i \in B, t_i \geq 0}{\text{minimum}}\, t_i.$$

Thus $\bar{t}$ corresponds to the first value of t with the property that $l_i < x_i(t) < u_i$ for some $i \in B$ and for each t near $\bar{t}$ with $t > \bar{t}$.

Iteration transitions

Now let us consider the transition between two consecutive major iterations. In making the transition between the final set $\overline{B}$ of the previous iteration and the initial set B_0 of the new iteration, rank-one corrections are both added and subtracted from the coefficient matrix $A_N Q_{NN}^{-1} A_N^T$. We now show that like the rank-one addition to $A_N Q_{NN}^{-1} A_N^T$, the rank-one subtractions can be treated in an incremental fashion.

Suppose that S and T satisfy the relation

$$T = S - UU^T$$

where U is arbitrary. The modification formula gives

$$T^{-1} = S^{-1} + S^{-1}U(I - U^T S^{-1} U)^{-1} U^T S^{-1}.$$

The following lemma gives a condition under which the matrix $I - U^T S^{-1} U$ is positive definite so that it can be stored in Cholesky factored form, and the factorization can be updated stably. This lemma clearly applies to the dual quadratic problems considered here where S and T have the form $A_N Q_{NN}^{-1} A_N^T$ for different choices of N. Moreover, when T is positive definite, any matrix of the form $U^T T^{-1} U$ is positive semidefinite, so that the hypotheses of the lemma will be satisfied.

LEMMA 2. *If $T = S - UU^T$ where S is symmetric and invertible, T is invertible, and the smallest eigenvalue of $U^T T^{-1} U$ is larger than -1, then $I - U^T S^{-1} U$ is positive definite.*

Proof. Observe that

$$I - U^T S^{-1} U = I - U^T (T + UU^T)^{-1} U.$$

By the modification formula,

$$(T + UU^T)^{-1} = T^{-1} - T^{-1} U (I + U^T T^{-1} U)^{-1} U^T T^{-1},$$

so we have

$$I - U^T S^{-1} U = I - U^T T^{-1} U + U^T T^{-1} U (I + U^T T^{-1} U)^{-1} U^T T^{-1} U.$$

If R denotes the product $U^T T^{-1} U$, then

$$I - U^T S^{-1} U = I - R + R(I + R)^{-1} R.$$

Hence, the eigenvectors of R and $I - U^T S^{-1} U$ are the same. If τ is an eigenvalue of R, then the associated eigenvalue of $I - U^T S^{-1} U$ is $1/(1 + \tau)$. Since $\tau > -1$, the matrix $I - U^T S^{-1} U$ is positive definite. □

Quadratic networks

Next, we formulate the important quadratic network problem, which has been considered by many researchers, (see [2], [3], [4], [5], [8], [9], [10], [12], [13], [14], [17], [26], [28], [29], [30], [32], [33], [34], [39], [40], [41], [49], [50] and [51]), and we describe an additional implementation idea that we call *null space steps*. The problem that we consider has the form

$$\text{minimize } f(x) \text{ subject to } Ax = b,\ l \leq x \leq u, \tag{19}$$

where

$$f(x) = \frac{1}{2}x^T Dx + c^T x. \tag{20}$$

Here x is a vector in $I\!R^n$; D is a diagonal matrix; l is a vector of lower bounds; u is a vector of upper bounds; $l < u$, b is a vector whose components sum to zero; and A is an $m \times n$ node-arc incidence matrix (see Strang [48]); that is, the number of nodes in the network is m, the number of arcs in the network is n, and if arc k connects node i to node j, then each element in column k of A is zero except for a -1 in row i and a $+1$ in row j. We assume that the diagonal of D is positive so that f is strictly convex. Since the rows of A sum to zero, we discard the last row to eliminate this dependency.

As in the prior development, the Lagrangian L is given by

$$L(\lambda, x) = f(x) + \lambda^T(Ax - b),$$

while the dual functional is

$$L(\lambda) = \text{ minimum } L(\lambda, x) \text{ subject to } l \leq x \leq u. \tag{21}$$

The dual problem is an unconstrained maximization:

$$\text{maximize } L(\lambda) \text{ subject to } \lambda \in I\!R^m. \tag{22}$$

By duality theory (see [35] or [44]), we know that if λ^* solves the dual problem and if the primal problem (19) has a solution x^*, then x^* attains the minimum in (21) corresponding to $\lambda = \lambda^*$.

Given a vector λ, let y denote the unconstrained minimizer of $L(\lambda, x)$ over all x. From the structure of f, y can be expressed as

$$y = -D^{-1}(c + A^T\lambda). \tag{23}$$

Let us partition the set $\{1, 2, \cdots, n\}$ into $B \cup N$ where

$$i \in N \text{ if } l_i < y_i < u_i, \text{ while } i \in B \text{ if } y_i \leq l_i \text{ or } y_i \geq u_i. \tag{24}$$

Due to the special form of f, the dual functional can be expressed

$$\begin{aligned} L(\lambda) = {} & \frac{1}{2}x_B^T D_{BB} x_B + c_B^T x_B + \lambda^T(A_B x_B - b) \\ & - \frac{1}{2}(c_N^T + \lambda^T A_N) D_{NN}^{-1} (c_N + A_N^T \lambda), \end{aligned} \tag{25}$$

where D_{BB} and D_{NN} are appropriate submatrices of D; and the ith component of x_B is equal to l_i if $y_i < l_i$, while it is equal to u_i if $y_i > u_i$. From (25) we see that L is a piecewise quadratic function. In addition, it is well known that a function like L, defined through minimization of a strictly convex quadratic, is continuously differentiable with

$$\nabla L(\lambda) = Ax(\lambda) - b$$

where $x = x(\lambda)$ achieves the minimum in (21) corresponding to λ.

Null space steps

If for some given λ and for some i, the y given by (23) has the property that $y_j < l_j$ or $y_j > u_j$ whenever $a_{ij} \neq 0$, then row i of A_N is completely zero, and both row i and column i of $A_N D_{NN}^{-1} A_N^T$ are zero. Since the quadratic terms in (25) associated with component i vanish, the function $L(\lambda)$ is a linear function of the ith component of λ. In other words, when all the primal variables associated with node i in the network are at their bounds, the only term in (25) involving λ_i is

$$\lambda_i(A_B x_B - b)_i.$$

When the coefficient of λ_i is nonzero, λ_i can be adjusted to strictly increase $L(\lambda)$, provided this is done without changing the set B in (24). A cycle through all the components of λ in which the value of those components associated with linearity is adjusted in order to increase L, while not changing the minimizer in (21), we refer to as a "null space step" since this adjustment takes place in the null space of the matrix $A_N D_{NN}^{-1} A_N^T$. In our implementation of the dual active set algorithm, the results of which appear in Section 6, each iteration begins with a null space step. Finite convergence of the method is not affected by a null space step since L strictly increases, and the proof of Theorem 1 is based on the fact that the final set $\overline{B}$ generated by the subiterations cannot repeat when L strictly increases.

5. Conjugate gradient techniques for quadratic networks

It has been shown by Lin and Pang [34] that gradient or conjugate gradient techniques can be used to maximize L (or M). In implementing one of these techniques, a line search is required. In our computational study of the quadratic network problem (19), we have included two versions of the nonlinear conjugate gradient algorithm. One of these versions uses the Polak-Ribière update formula [46] as given in Luenberger [35, p. 253], and the other adds a preconditioner. Ventura and Hearn [50] found that the Polak-Ribière conjugate gradient update is more efficient than others in numerical experiments involving networks. This section describes our line search procedure and the preconditioner.

We consider an exact line search, although global convergence can be achieved using an inexact line search (see [22] for example). If λ_k is the current iterate and p_k is the current search direction, then with exact line search, we have

$$\lambda_{k+1} = \lambda_k + s_k p_k,$$

where the step size s_k is given by

$$s_k = \arg\max_{s \geq 0} L(\lambda_k + s p_k). \tag{26}$$

Let g_k denote the gradient $\nabla L(\lambda_k)$, and let us assume that $p_k^T g_k > 0$ so that p_k is a direction of ascent. At a maximum in (26), the derivative vanishes, and we have

$$p_k^T(Ax(s_k) - b) = 0,$$

where $x(s)$ is the minimizer in (21) corresponding to $\lambda = \lambda_k + s p_k$. Let $q(s)$ denote the expression $p_k^T(Ax(s) - b)$. The problem of determining a root of the equation

$$q(s) = 0 \tag{27}$$

has been studied extensively in the literature (see [6], [27], and [42]). The following proposition shows that this equation will have a root unless the primal problem is infeasible:

PROPOSITION. *The function $q(s)$ is a continuous, piecewise linear, monotone nonincreasing function of s. If $p_k^T g_k > 0$, then either the equation $q(s) = 0$ has a root, or the primal problem* (19) *is infeasible, and L grows without bound in the direction p_k.*

Proof. The fact that $q(s)$ is continuous, piecewise linear, monotone nonincreasing follows from the formula (23) for the unconstrained minimizer associated with (21) (see [27]). Suppose that (27) does not have a root. Since $p_k^T g_k > 0$, we conclude that $q(0) > 0$ and $q(s) > 0$ for every $s \geq 0$. If $y(s)$ is defined by

$$y(s) = D^{-1}\left[c + A^T(\lambda_k + s p_k)\right],$$

then $x_i(s) = y_i(s)$ if $l_i \leq y_i(s) \leq u_i$, $x_i(s) = l_i$ if $y_i(s) < l_i$, and $x_i(s) = u_i$ if $y_i(s) > u_i$. From the construction of $x(s)$, we see that there exists t such that $x(s) = x(t)$ for $s \geq t$. Hence, $q(s) = q(t)$ for $s \geq t$, from which it follows that

$$L(\lambda_k + sp_k) = f(x(t)) + \lambda_k^T(Ax(t) - b) + sq(t) \text{ for } s \geq t.$$

Since $q(t) > 0$, the maximum in (22) is infinite. Since the minimum in the primal problem (19) bounds the maximum in the dual problem, the primal problem is infeasible. □

We now make a few practical observations which provide the basis for the line search routine used in the numerical experiments reported in Section 6. The function q is monotone nonincreasing with a finite number of "kinks" where the slope changes. Each kink has an associated "breakpoint" on the axis. A Newton iteration, starting from any point where q is positive (such as $s = 0$), generates a point to the right. (If an iterate is a breakpoint of q, then either slope of the adjacent linear segments of q is used in the Newton iteration.) There are three possibilities for the resulting Newton iterate: (i) It is a zero of q (in which case the iteration terminates); (ii) it lies to the right of a zero; or (iii) it lies to the left of a zero, to the right of the starting point, and at a point corresponding to a different linear segment on the graph of q. In case (iii), we can continue to apply Newton's method, and in a finite number of iterations, either a zero of q is generated, or a point to the right of a zero is found. In summary, after a finite number of Newton iterations, starting from $s = 0$, we either locate or bracket a zero of q. Once a zero is bracketed, we sort the bracketed breakpoints. Starting at one bracketing point (we use the point where the absolute value of q is smallest), we march to the other bracketing point to determine a pair of breakpoints where the graph of q crosses the axis. In Figure 1, the line search algorithm, starting from s_0, generates s_1 and s_2 by Newton steps, thereby bracketing the zero of $q(s)$. It then calculates and sorts the four breakpoints between s_1 and s_2, and locates the zero between the first and second of these. We have observed in our numerical experiments connected with the conjugate gradient method, that 1 or 2 Newton iterations typically generate a point that brackets a zero of q. Moreover, the number of bracketed break points is often a small fraction of n, and the time to sort these points is relatively insignificant.

Now consider the problem of finding the step size s_k that maximizes M in the search direction p_k:

$$s_k = \arg\max_{s \geq 0} M(\lambda_k + sp_k).$$

Again, let us assume that p_k is a direction of ascent. Since the derivative vanishes at the maximum, we have

$$p_k^T(Ax(s_k) - b) - 2\varepsilon p_k^T(\lambda_k + s_k p_k - \Lambda) = 0.$$

Let $r(s)$ denote the expression $q(s) - 2\varepsilon p_k^T(\lambda_k + sp_k - \Lambda)$. Since $q(s)$ is continuous, piecewise linear, monotone nonincreasing, it follows that $r(s)$ is continuous, piecewise linear, strictly decreasing. Consequently, the algorithm described above for finding a zero of q can be used to find a zero of r.

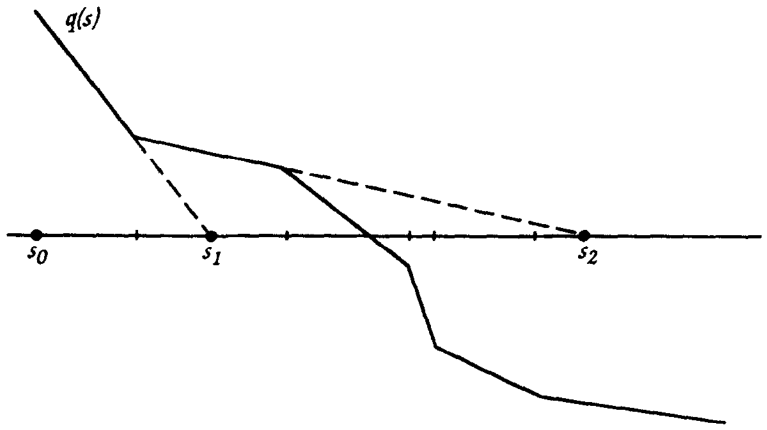

Figure 1. Line search example.

Our preconditioned version of the conjugate gradient method is motivated by the dual active set algorithm. Recall from the previous section that the dual active set algorithm successively adds rank-one corrections to the matrix associated with the quadratic part of L_B^z as additional primal variables are freed. This suggests a related diagonal preconditioner for the conjugate gradient method in which we successively add the diagonal part of the rank-one corrections as primal variables are freed. The preconditioned conjugate gradient method (see [21]) has the following form:

$$\lambda_{k+1} = \lambda_k + s_k p_k,$$
$$p_{k+1} = P^{-1} g_{k+1} + \beta_k p_k,$$

where g_k is the gradient of the cost function L, evaluated at λ_k; and P is the preconditioner. For the Polak-Ribière update, we have

$$\beta_k = \frac{g_{k+1}^T P^{-1}(g_{k+1} - g_k)}{g_k^T P^{-1} g_k}.$$

Observe that the matrix associated with the quadratic part of L in (25) has the following form:

$$\frac{1}{2} \sum_{i \in N} \frac{1}{d_i} a_i a_i^T$$

where d_i denotes the ith diagonal element of D; and a_i denotes the ith column of A. In iteration k of the preconditioned conjugate gradient method, we utilize a diagonal preconditioner whose diagonal has the form

$$\text{diagonal } P = \frac{1}{2}\sum_{i\in F}\frac{1}{d_i} \text{ diagonal } \{a_i a_i^T\},$$

where the initial F is given by

$$F = \{i : l_i \leq y(\lambda_0)_i \leq u_i\}. \tag{28}$$

In each preconditioned iteration, the gradient is multiplied by the "inverse" of P. Since some diagonal elements of P may be zero, our convention is that $0^{-1} = 1$. The set F is updated in the following way: If $y(\lambda_k)_i < l_i$ or $y(\lambda_k)_i > u_i$ whenever $i \notin F$, then F does not change. Otherwise, F is augmented:

$$F^{\text{new}} = F^{\text{old}} \cup \{i \notin F^{\text{old}} : l_i \leq y(\lambda_k)_i \leq u_i\},$$

and the conjugate gradient iteration is restarted using this new F for the preconditioner. If F remains invariant for some preset number of iterations, then the conjugate gradient scheme is restarted and the set F is reinitialized using (28). In our numerical experiments, a restart was performed after m iterations, where m is the number of nodes in the network.

6. Numerical experiments

The numerical performance of the dual active set algorithm was evaluated using 16 of the quadratic network test problems studied by Bertsekas et al. [5] as well as a test problem considered by Toint and Tuyttens [49] (only one of their test problems, $P(8, 0, 10^2, 1, \frac{1}{10})$, had a quadratic cost). These experiments were performed using an Apollo DN10000 computer and the Fortran f77 compiler without optimization. All computing times reported below are in seconds. The starting guess was always $\lambda_0 = 0$. Each algorithm was terminated when $\|\nabla L(\lambda_k)\|/\|b\| \leq 10^{-6}$. Since $\nabla L(\lambda_k) = Ax_k - b$, where x_k achieves the minimum in (21) corresponding to $\lambda = \lambda_k$, the termination criterion is equivalent to a bound on the relative residual associated with the linear system $Ax = b$.

The test problems in [5] are randomly generated networks constructed using Zenios' modification of the NETGEN program [31]. This modification generates a quadratic term for the cost function as well as a linear term. The test problems that we considered are summarized in Table 1. As in [5], problems 1–8 are constructed using the code of Zenios while problem 9 is $P(8, 0, 10^2, 1, \frac{1}{10})$ from [49]. We considered the same two versions of the test problems studied in [5]; a well-conditioned version where the coefficients of the quadratic part range between 5 and 10, and an ill-conditioned version where 50% of the arcs are randomly set to a small positive number that appears in the fourth column of Table 1. The coefficients for problem 9 range between 0.02 and about 3.86.

To begin, we examine the efficiency of the conjugate gradient method on well-conditioned problems relative to a dual coordinate ascent code discussed

Table 1. Test problem summary.

Problem Number	Number of Nodes	Number of Arcs	Small Quadratic Coefficient
1	200	1300	.0001
2	200	2900	.001
3	300	4500	.0001
4	400	1500	.0001
5	400	4500	.01
6	400	1306	.001
7	400	1306	.001
8	400	1382	.001
9	324	612	

Table 2. Execution times for well-conditioned test problems.

Problem Number	Dual Coordinate Ascent	Unconditioned CG	Preconditioned CG
1	1.72	1.63	1.52
2	4.22	4.77	4.22
3	4.23	4.28	3.53
4	3.38	2.17	1.75
5	5.40	7.32	6.45
6	3.48	2.62	3.07
7	3.42	2.03	1.93
8	2.95	2.27	1.72
9	76.73	5.42	4.35

in [5]. Table 2 gives the execution times for the nine test problems. The total execution times for well-conditioned problems 1–8 is 28.80 (dual coordinate ascent), 27.09 (unconditioned conjugate gradients), and 24.19 (preconditioned conjugate gradients). Thus, the codes are somewhat comparable on the first eight problems. On problem 9, which was not considered in [5], the conjugate gradient method is significantly faster.

Now consider the ill-conditioned versions of problems 1–8. Similar to the observations reported in [5], we found that the dual coordinate ascent code did not achieve more than a few digits of accuracy so that the convergence criterion $\|\nabla L(\lambda_k)\|/\|b\| \le 10^{-6}$ could not be satisfied. The conjugate gradient method converged for the ill-conditioned problems, but the computing times were relatively large. The times in Table 3 correspond to a restart of the conjugate gradient

Table 3. Execution times for ill-conditioned test problems.

Problem Number	Unconditioned CG	Preconditioned CG	Active Set Scheme Alone	Active Set Scheme With CG
1	217.97	159.90	42.43	8.07
2	100.68	87.35	67.30	18.25
3	667.30	378.35	226.93	35.95
4	544.13	329.88	94.50	19.37
5	61.98	49.70	188.52	26.13
6	173.57	100.87	75.92	15.18
7	121.00	123.28	74.00	14.05
8	106.62	88.20	86.40	16.75
Total	1993.25	1317.53	856.00	153.75

method every m iterations. The total execution times for ill-conditioned problems 1–8 is 1,993.25 (unconditioned) and 1,317.53 (preconditioned), more than 50 times slower than the corresponding times for the well-conditioned problems.

In implementing the dual active set algorithm, we factored the coefficient matrix of the associated linear system at the start of each iteration using the minimum degree ordering routine of Sparspak [16]. During the subiterations, the associated linear systems were solved using Sherman-Morrison-Woodbury updating techniques as explained in Section 4. We chose $\varepsilon_0 = 0.001$ so that the initial value of the regularization parameter is about 100 times smaller than the smallest nonzero coefficient in the linear system associated with the dual iteration. In successive iterations of the underlying proximal point iteration discussed in Section 3, we took $\varepsilon_{k+1} = 0.1\varepsilon_k$. Since the rows of A sum to zero, we eliminated the last row, and we normalized the dual variable so that $\lambda_m = 0$. (It was observed that eliminating a redundant row of A and normalizing the dual variable slows the convergence of the conjugate gradient method; consequently, this reduction was not used with the conjugate gradient method.)

By itself, the dual active set algorithm is more efficient than preconditioned conjugate gradient iterations—see column 4 of Table 3. The total execution time is 856 (dual active set algorithm) compared to 1,317.53 (preconditioned conjugate gradients). Moreover, if we switch to the dual active set algorithm after performing a small number of conjugate gradient iterations, then there is a substantial speedup. The execution times in column 5 of Table 3 correspond to $0.3m$ unconditioned conjugate gradient iterations, a number determined experimentally on problem 1 to be the best point at which to switch to the dual active set algorithm. The total time in column 5 is 153.75, around an 8.5-fold speedup relative to preconditioned conjugate gradients alone, and a 13-fold speedup relative to unconditioned conjugate gradients. Another attractive feature of the dual active set algorithm, besides its speed near an optimum, is that it generates

Table 4. Number of subiterations for ill-conditioned test problems.

Iteration	Problem							
Number	1	2	3	4	5	6	7	8
1	133	172	231	183	68	173	204	159
2	27	43	44	49	2	21	26	20
3	11	34	25	5	1*	8	8	8
4	3	8	18	1	1	3	1*	1
5	2*	3*	5	1*		3	2	1*
6	2	1	7	4		1*	1*	3
7			1	5		3	1	4*
8			1*	1		2		1
9			1	1*		1*		
10				1		2		
11								

very accurate solutions—after a few iterations (typically between 5 and 15), the algorithm reaches nearly the exact solution. The number of subiterations associated with the combined conjugate gradient/dual active set algorithm appear in Table 4. The iterations where the solution to the proximal point subproblem are achieved (and where ε_k is multiplied by 0.1) are flagged with an asterisk.

We also investigated a slightly different implementation of the proximal point iteration in which the regularization parameter was kept fixed at $\varepsilon_k = 0.001$, while the true maximizer of the proximal point function M was replaced by an approximation computed by applying one iteration of the dual active set algorithm. In other words, λ_{k+1} was obtained by applying one iteration of the dual active set algorithm to the function

$$L(\lambda) - .001\|\lambda - \lambda_k\|^2.$$

We observed that the total computing time for problems 1–8 was reduced from 153.75 to 144.25.

To compare the dual active set algorithm to the algorithms investigated in [49], we solved problem 9 using the same type computer (VAX 8600) and the same compiler (VMS Fortran Compiler with optimization) employed in [49]. The computing time that we obtained was 7.97 seconds (for the original implementation of the proximal point iteration and with $0.3m$ unconditioned conjugate gradient iterations before switching to the dual active set algorithm), more than a 10-fold speedup over the times reported in Table 4 of [49]. Note, however, that the code in [49] is designed for general nonlinear networks, while our code is tailored to quadratic networks.

In summary, the dual active set algorithm is a highly effective method for quadratic network flow problems, especially when used in conjunction with the nonlinear conjugate gradient method.

Acknowledgment

The authors thank D.P. Bertsekas and P. Tseng for providing a copy of their dual coordinate ascent code. The authors thank H-D. Chen for his assistance with the numerical experiments.

This research was supported by the U.S. Army Research Office contract DAAL 03-89-G-0082, the National Science Foundation grant DDM-8814075, and the Southwest Florida Water Management District.

Appendix. A Matlab program

This appendix provides a Matlab implementation of the dual active set algorithm for a quadratic program in which the matrix associated with the quadratic part is diagonal with positive diagonal elements.

```
% dasa solves the separable strictly convex quadratic program:
%  minimize f(x) = 0.5 x'Dx + c'x subject to Ax = b, l <= x <= u
% by the dual active set algorithm.
% Reference to this function is made by:
% [f, x, lambda] = dasa(D, c, A, b, l, u, Lambda)
% where D, c, A, b, l, and u are the problem data and Lambda is an
% initial guess for the dual variables associated with Ax = b,
% f is the final objective value, and x and lambda are the final primal
% and dual variables.
%
function [f, x, lambda]=dasa(D, c, A, b, l, u, Lambda)
   [m, n] = size(A);
   x = 0 * c;                          %initialize x as a column vector
% Set tolerances for each loop
   tol2 = norm(b) * 10^(-6); tol3 = tol2;
   lambda = Lambda; epsilon = 0.01;  % epsilon used in proximal point iterations

% proximal point outer loop  -loop3- - - - - - - - - - - - - - - - - - - - - -

 loop3 = 1;
 while loop3,
    epsilon = 0.1 * epsilon;
%  dual iteration middle loop  -loop2- - - - - - - - - - - - - - - - - -

    loop2 = 1;
    while loop2,
      z = -D \ (c + A' * lambda);  % unrestricted min of the Lagrangian
      B = zeros(c);   % B will index the variables at bounds
      for i = l : n     % set variables to their bounds
          if  z(i) <= l(i), B(i) = -1; z(i) = l(i);
          elseif z(i) >= u(i), B(i) = 1; z(i) = u(i); end
      end
%       Dual subiteration inner loop  -loop1- - - - - - - - - - -

     loop1 = 1; v = lambda;  % v = dual subiterates
     while loop1,
      % - - - solve for mu - - -
      fr = find(B == 0); bd = find(B~= 0); % indices of free and bound variables
      if length(fr) == 0;
          mu = Lambda + (A * z - b)/(2 * epsilon);
          x = z;
      else
          DN = D(fr, fr); AN = A(:,fr);
          Q = AN * inv(DN) * AN' + 2 * epsilon * eye(m):
          q = -b + 2 * epsilon * Lambda - AN * (DN\c(fr));
          if length(bd) > 0, q = q + A(:,bd) * z(bd); end
          mu = Q \ q;
          % find x at the point mu
          x(fr) = -DN\(AN' * mu + c(fr));
          if length(bd) > 0, x(bd) = z(bd); end
      end
      % - - - linear search for tbar - - -
      %  t contains the stepsizes where partials equal 0
          num = -(D\(c + A' * v) + z);
          denom = (D\A' * (mu - v)));
          t = (1e20) * ones(n, 1);
          for i = l : n
            if denom(i)~= 0, t(i) =num(i)/denom(i);end
          end;
          % find tbar
          indices = find(t >= 0 & B~= 0);
          if length(indices) == 0, loop1 = 0;
           else [tbar,i] = min(t(indices));
              B(indices(i)) = 0;
              mutbar = v+tbar * (mu - v);
          end
          if tbar > = 1, loop1 = 0, else v = mutbar; end
     end % - - - - - - - - - - - - - - - - - - - - - end of loop1
```

```
            lambda = mu;
            if norm($A*x-b-2*$ epsilon $*$ (lambda − Lambda))<tol2, loop2=0; end
        end % - - - - - - - - - - - - - - - - - - - - - - end of loop 2
        Lambda = lambda;
        if norm($A*x-b$) < tol3, loop3 = 0; end
    end % - - - - - - - - - - - - - - - - - - - - - - end of loop3
    lambda = Lambda;
    $f = c'*x+0.5*x'*D*x$;
```

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley: Reading, MA, 1974.
2. A. Bachem and B. Korte, "Minimum norm problems over transportation polytopes," *Linear Algebra and Its Applications* **31** (1980), 103–118.
3. P. Beck, L. Lasdon, and M. Engquist, "A reduced gradient algorithm for nonlinear network problems," *ACM Trans. on Math. Software* **9** (1983), 57–70.
4. D.P. Bertsekas and D. El Baz, "Distributed asynchronous relaxation methods for convex network flow problems," *SIAM J. on Control and Optimization* **25** (1987), 74–85.
5. D.P. Bertsekas, P.A. Hosein, and P. Tseng, "Relaxation methods for network flow problems with convex arc costs," *SIAM J. on Control and Optimization* **25** (1987), 1219–1243.
6. G.R. Bitran and A.C. Hax, "Disaggregation and resource allocation using convex knapsack problems with bounded variables," *Mgt. Sci.* **27** (1981), 431–441.
7. F.H. Clarke, "Generalized gradients and applications," *Trans. of the American Mathematical Society* **205** (1975), 247–262.
8. M. Collins, L. Copper, R. Helgason, J. Kennington, and L. LeBlanc, "Solving the pipe network analysis problem using optimization techniques," *Mgt. Sci.* **24** (1978), 747–760.
9. L. Cooper and J. Kennington, "Steady state analysis of nonlinear resistive electrical networks using optimization techniques," Tech. Report IEOR 77012, Southern Methodist University, Dallas, TX, 1977.
10. R.W. Cottle, S.G. Duvall, and K. Zikan, "A Lagrangean relaxation algorithm for the constrained matrix problem," *Nav. Res. Logistics Q.* **33** (1986), 55–76.
11. L.H. Cox, "Solving statistical confidentiality problems via network optimization," TIMS/ORSA Joint National Meeting, New Orleans, LA, 1987.
12. J.L. Debiesse and G. Matignon, "Comparison of different methods for the calculation of traffic matrices," *Annales des Telecommunications* **35** (1980), 91–102.
13. R.S. Dembo and J.G. Klincewicz, "A scaled reduced gradient algorithm for network flow problems with convex separable costs," *Math. Programming Study* **15** (1981), 124–147.
14. R.S. Dembo, "A primal truncated Newton algorithm with application to large-scale nonlinear network optimization," *Math. Programming Study* **31** (1987), 43–72.
15. M.E. El-Hawary and G.S. Christensen, *Optimal economic operation of electric power systems*, Academic Press: New York, NY, 1979.
16. A. George and J.W.H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall: Englewood Cliffs, NJ, 1981.
17. C.R. Glassey, "A quadratic network optimization model for equilibrium of single commodity trade flows," *Math. Programming* **14** (1978), 98–107.
18. Goldfarb and Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Math. Programming* **27** (1983), 1–33.

19. C.D. Ha, "A generalization of the proximal point algorithm," *SIAM J. on Control and Optimization* **28** (1990), 503–512.
20. W.W. Hager, "Inequalities and approximation," in *Constructive Approaches to Mathematical Models*, C.V. Coffman and G.J. Fix, eds., Academic Press: New York, NY, 1979 189–202.
21. W.W. Hager, "Dual techniques for constrained optimization," *J. of Optimization Theory and Applications* **55** (1987), 37–71.
22. W.W. Hager, "A derivative-based bracketing scheme for univariate minimization and the conjugate gradient method," *Computers and Math. with Applications* **18** (1989), 779–795.
23. W.W. Hager, "Updating the inverse of a matrix," *SIAM Review* **31** (1989), 221–239.
24. W.W. Hager, "The dual active set algorithm," in *Advances in Optimization and Parallel Computing*, P.M. Pardalos, ed., North Holland: Amsterdam, The Netherlands, 1992, 137–142.
25. W.W. Hager and G.D. Ianculescu, "Dual approximations in optimal control," *SIAM J. on Control and Optimization* **22** (1984), 423–465.
26. D.W. Hearn, S. Lawphongpanich, and J.A. Ventura, "Restricted simplicial decomposition: Computation and extensions," *Math. Programming Study* **31** (1987), 99–118.
27. R.V. Helgason, J. L. Kennington, and H. Lall, "Polynomially bounded algorithm for a single constrained quadratic program," *Math. Programming* **18** (1980), 338–343.
28. P.V. Kamesam and R.R. Meyer, "Multipoint methods for nonlinear networks," *Math. Programming Study* **22** (1984), 185–205.
29. J.G. Klincewicz, "A Newton method for convex separable network flow problems," *Networks* **13** (1983), 427–442.
30. J.G. Klincewicz, "Implementing an 'exact' Newton method for separable convex transportation problems," *Networks* **19** (1989).
31. K. Klingman, A. Napier, and J. Stutz, "NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems," *Mgt. Sci.* **20** (1974), 814–821.
32. L.J. LeBlanc, R.V. Helgason, and D.E. Boyce, "Improved efficiency of the Frank-Wolfe algorithm for convex network programs," *Transp. Sci.* **19** (1985), 445–462.
33. L.J. LeBlanc, "The conjugate gradient technique for certain quadratic network problems," *Nav. Res. Logistics Q.* **23** (1976), 597–602.
34. Y.Y. Lin and J.S. Pang, "Iterative methods for large convex quadratic programs: A survey," *SIAM J. on Control and Optimization* **25** (1987), 383–441.
35. D.G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley: Reading MA, 1984.
36. F.J. Luque, "Asymptotic convergence analysis of the proximal point algorithm," *SIAM J. on Control and Optimization* **22** (1984), 277–293.
37. B. Martinet, "Regularisation d'inèquations variationelles par approximations successives," *Revue Francaise Informatique et Recherche Opérationnelle* (1970), 154–159.
38. B. Martinet, "Determination approachée d'un point fixe d'une application pseudocontractante," *Comptes Rendus des Séances de l'Académie des Sciences, Paris* **274** (1972), 163–165.
39. M. Minoux, "A polynomial algorithm for minimum quadratic cost flow problems," *European J. of Operational Res.* **18** (1984), 377–387.
40. J.M. Mulvey, S.A. Zenios, and D.P. Ahlfeld, "Simplicial decomposition for convex generalized networks," Research Report No. EES-85-8, Civil Engineering Department, Princeton University, Princeton, NJ, 1985.
41. A. Ohuchi and I. Kaji, "Lagrangian dual coordinatewise maximization algorithm for network transportation problems with quadratic costs," *Networks* **14** (1984), 515–530.
42. P.M. Pardalos and N. Kovoor, "An algorithm for singly constrained quadratic programs," *Math. Programming* **46** (1990), 321–328.
43. E. Polak and G. Ribière, "Note sur la convergence de methods de directions conjugres," *Revue Francaise Informatique et Recherche Opèrationnelle* **16** (1969), 35–43.
44. R.T. Rockafellar, *Convex Analysis*, Princeton University Press: Princeton, NJ, 1970.

45. R.T. Rockafellar, "Augmented Lagrangians and applications of the proximal point algorithm in convex programming," *Math. of Operations Res.* **2** (1976), 97–116.
46. R.T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM J. on Control and Optimization* **14** (1976), 877–898.
47. J.E. Spingarn, "Submonotone mappings and the proximal point algorithm," *Numerical Functional Analysis and Optimization* **4** (1982), 123–150.
48. G. Strang, *Introduction to Applied Mathematics,* Wellesley-Cambridge Press: Wellesley, MA, 1986.
49. P.L. Toint and D. Tuyttens, "On large scale nonlinear network optimization." *Math. Programming* **48** (1990), 125–159.
50. J.A. Ventura and D.W. Hearn, "Computational development of a Lagrangian dual approach for quadratic networks," Industrial and Systems Engineering Department, Report 87–8, University of Florida, Gainesville, FL, 1987.
51. S.A. Zenios and J.M. Mulvey, "Relaxation techniques for strictly convex network problems," *Annals of Operations Research* **5** (1985/6), 517–538.