

Quadratic Cost Flow and the Conjugate Gradient Method*

Jie Sun[†]

School of Business and Singapore-MIT Alliance
National University of Singapore, Republic of Singapore

Xiaoqi Yang[‡]

Department of Mathematics, Hong Kong Polytechnic University, Hong Kong.

Xiongda Chen[§]

Department of Applied Mathematics, Shanghai Tongji University, China

April 23, 2003

Abstract

By introducing quadratic penalty terms, a convex nonseparable quadratic network program can be reduced to an unconstrained optimization problem whose objective function is a piecewise quadratic and continuously differentiable function. A conjugate gradient method is applied to the reduced problem and its convergence is proved. The computation exploits the special network data structures originated from the network simplex method. This algorithmic framework allows direct extension to multicommodity cost flows. Some preliminary computational results are presented.

Keywords: Conjugate gradient methods, network quadratic programming.

*The research was partially supported by Grants R314-000-028/042-112 of National University of Singapore and a grant from Singapore-MIT Alliance.

[†]E-mail: jsun@nus.edu.sg Fax: 65-67792621, Address: 02-16, 1 Business Link, Singapore 117592.

[‡]E-mail: mayangxq@polyu.edu.hk

[§]E-mail: chenxiongda@yahoo.com

1 Introduction

Conjugate gradient methods has long been used for solving large-scale unconstrained optimization problems. In particular, it is shown that if the objective is convex quadratic, then the method will solve the problem in n iterations, where n is the dimension of the variable.

A piecewise quadratic function from \mathcal{R}^n to \mathcal{R} is a continuous function $p(\cdot)$ such that

$$p(x) \in \{q_1(x), \dots, q_K(x)\} \quad \forall x \in \mathcal{R}^n,$$

where $q_i(\cdot)$, $i = 1, \dots, K$, is quadratic. We include the case where some $q_i(\cdot)$ s may be affine functions. If $p(\cdot)$ is convex, then the requirement for continuity could be dropped since any convex function in \mathcal{R}^n is necessarily continuous [7]. The minimization of a piecewise quadratic function arises, for example, when some constraints are treated as quadratic penalty terms in the constrained optimization. Even if the constraints are treated as linear penalties, one still ends up with a piecewise quadratic objective function if the original objective function is quadratic. Another important source of piecewise quadratic functions is stochastic programming, where p is the optimal value function of a recourse action [4].

If we apply a conjugate gradient method to minimizing a convex piecewise quadratic function, then we can not expect finite convergence in general unless the minimum point is in the interior of a “piece”, i.e., there exists a neighborhood of the minimum point x^* such that on the entire neighborhood $p(x) = q_j(x)$ for some fixed $j \in \{1, \dots, K\}$. However, even without finite convergence, the conjugate gradient method is still valuable for large-scale problems since it does not require to compute any matrix inverse. Particularly in the network case, we shall show that the conjugate gradient method can take advantage of the network data structure to greatly reduce the amount of computation. Therefore, the conjugate gradient method might be a good choice for network nonlinear programs, where the problems are so large that none of the current favorite nonlinear programming methods, such as Newton methods or trust region methods, could hopefully solve them.

Our specific interest in this paper is originated from solving the convex network quadratic program

$$\text{minimize } f(x) = \frac{1}{2}x^\top Qx + c^\top x \quad \text{subject to } Ex = s, \quad l \leq x \leq u. \quad (\text{NQP})$$

The problem is very important, which for instance may arise as a subproblem in various algorithms for more complicated nonlinear network optimization problems. One of such examples – the multicommodity quadratic flow problem – will be discussed briefly at the end of this paper.

We do not assume the problem (NQP) to be separable, namely we do not require the matrix Q in the objective function to be diagonal. However, for ease of discussion, we assume that Q is symmetric positive semidefinite, although a more sophisticated analysis could be done without this assumption.

Our idea of solving problem (NQP) is to incorporate the inequality constraints into the objective function so as to reduce the problem into an equality constrained problem that will allow us to take full advantage of the network structure. We use a quadratic penalty function, which of course is an inexact penalty function and makes the objective function piecewise quadratic. It can be shown [9, Theorem 2.1] that the solution of such penalized problem can approximate the real solution of (NQP) to any prespecified tolerance. For practical purposes, this type of solution should be good enough. The gain from the quadratic penalty function is that we have a continuously differentiable (i.e. C^1) objective function in the penalized problem that will allow the conjugate gradient method to converge, as will be shown in Section 2. A similar approach has been used by Sun and Kuo [9] in a Newton’s method for strictly convex separable network quadratic programs. Compared to [9], our method applies to nonseparable (non-strictly) convex problems and does not involve matrix inverse operations.

This paper is organized as follows. We give a complete description of the problem and introduce the conjugate gradient method for piecewise quadratic programs in Section 2. In Section 3 we state the reduced conjugate gradient method applied to equality constrained problem and provide some implementation schemes. Some numerical results are reported in Section 4. We make some concluding remarks in Section 5.

2 The problem

2.1 Problem formulation

A directed network consists of m nodes and n arcs. Each arc j originates at some node i and terminates at another node i' . In a network flow model a certain commodity travels along the arcs in order to satisfy the supply or demand requirements at the nodes. It is convenient to think of demand as negative supply in this case. Let s_i , $i = 1, \dots, m$, be the (positive or negative) supply at node i , and let the total amount of the commodity that travels along a particular arc j be x_j , $j = 1, \dots, n$. Following Rockafellar [8], we call x_j the *flux* on arc j and call

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \text{and} \quad s = \begin{pmatrix} s_1 \\ \vdots \\ s_m \end{pmatrix}$$

the *flow vector* and the *supply vector* respectively. We define the node-arc incidence matrix by

$$E = \{e(i, j)\}_{i=1}^m \{j=1}^n$$

where

$$e(i, j) = \begin{cases} +1 & \text{if } i \text{ is the initial node of the arc } j, \\ -1 & \text{if } i \text{ is the terminal node of the arc } j, \\ 0 & \text{otherwise.} \end{cases}$$

Note that by the definitions of E and x the i th entry of the vector Ex represents the total supply of x at node i . Thus, (NQP) seeks to find the best flow vector x to minimize the total quadratic costs incurred on the arcs, subject to the outside supply requirements at the nodes as well as bound constraints for the flow vector.

It is well known in the literature that the system $Ex = s$ can be equivalently transformed into $x_B = Rx_N + b$, where x_B is the basic variables and x_N are the nonbasic variables. To do so, one can find that $E = [E_B, E_N]$ with E_B invertible, and let $R = -E_B^{-1}E_N$ and $b = E_B^{-1}s$. The arcs corresponding to x_B form a maximal spanning tree of the network. The matrix R can then be realized and stored in terms of the spanning tree. Any computations involving the forms of Rv and $R^\top u$ can be implemented in an extremely efficient way due to the data structures originated from the network simplex method. We will see later in this paper that in our implementation of the conjugate gradient method, exactly one matrix-vector multiplication of the form $R^\top u$ and one of the form Rv are necessary at each iteration.

2.2 The penalty terms

Consider the penalized network quadratic program

$$\begin{aligned} & \text{minimize} && p(x) \equiv f(x) + \frac{\theta}{2} \sum_{j=1}^n [\min^2\{x_j - l_j, 0\} + \min^2\{u_j - x_j, 0\}] \\ & \text{subject to} && Ex = s. \end{aligned} \tag{PNQP}$$

This problem is obtained by dropping the inequality constraints $l \leq x \leq u$, (where l and u are the given lower and upper bound vectors, respectively). If there is no lower or upper bound for some flux x_j , the corresponding term will not show up in the objective function) and adding the penalty term $\frac{\theta}{2} \sum_{j=1}^n [\min^2\{x_j - l_j, 0\} + \min^2\{u_j - x_j, 0\}]$ in (NQP), where $\theta > 0$ is the penalty parameter. Obviously, $p(\cdot)$ is convex if and only if $f(\cdot)$ is. In addition, $p(\cdot)$ is continuously differentiable.

For $\delta \geq 0$, a solution x is δ -feasible to (NQP) if $Ex = s$ and $\min_j\{x_j - l_j, u_j - x_j\} \geq -\delta$. A δ -feasible solution is ε -optimal to (NQP) if $f(x)$ is within ε of the optimal value of (NQP). In practice we start from an experimental value of θ , say 10 or 100. If the solution is not δ -feasible and ε -optimal to (NQP), in the next iteration we replace θ with 10θ and repeat the algorithm until the solution is δ -feasible and ε -optimal to (NQP) or the problem is proven to be unbounded. This scheme is quite similar to that suggested by

Gamble, Conn, and Pulleyblank [3] except that in [3], the objective function is linear and an exact penalty function is used, which yields the exact optimal solution ($\delta = \epsilon = 0$). The following theorem ensures that the process of enlarging θ will end up with a δ -feasible and ϵ -optimal solution in finitely many iterations. The convex separable case of the theorem was proved in [9, Theorem 2.1]. It is straightforward to extend the proof to the non-separable case. For brevity, we omit the proof.

Theorem 2.1 *Suppose that problem (NQP) has a solution. For any $\delta > 0$ and $\epsilon > 0$ there exists a $\bar{\theta} > 0$ such that for $\theta > \bar{\theta}$ there is an optimal solution of (PNQP) which is also a δ -feasible and ϵ -optimal solution of (NQP).*

2.3 The conjugate gradient method

The conjugate gradient method is quite useful in finding an unconstrained minimum of a high-dimensional function $F : \mathcal{R}^n \rightarrow \mathcal{R}$. In general, the method has the following form:

$$d^k = \begin{cases} -g^k, & \text{for } k = 1, \\ -g^k + \beta_k d^{k-1}, & \text{for } k > 1, \end{cases} \quad (1)$$

and

$$x^{k+1} = x^k + \alpha_k d^k, \quad (2)$$

where g^k denotes the gradient $\nabla F(x^k)$, d^k is the search direction, α_k is a steplength obtained by a line search, and β_k is chosen so that d^k becomes the k th conjugate direction when the function is quadratic and the line search is exact. A well-known formula for β_k is given by

$$\beta_k = \frac{\|g^k\|^2}{\|g^{k-1}\|^2}, \quad (\text{Fletcher-Reeves [2]}). \quad (3)$$

In general, a conjugate gradient method may not be convergent if applied to a general convex function. However, in the case where $F(x)$ is a piecewise quadratic convex C^1 function, the global convergence of the conjugate gradient method (1)-(3) is shown in the following theorem.

Theorem 2.2 *Let $F : \mathcal{R}^n \rightarrow \mathcal{R}$ be C^1 -piecewise quadratic and convex. Suppose that the line search is performed exactly, i.e. $\alpha_k = \operatorname{argmin}_{\alpha \geq 0} F(x^k + \alpha d^k)$. Then starting from any x^1 , the conjugate gradient algorithm defined by (1)-(3) generates a sequence $\{x^k, k = 1, 2, \dots\}$ such that either $\lim F(x^k) = -\infty$ or $\liminf \nabla F(x^k) = 0$.*

Proof. Suppose that $\mathcal{R}^n = P_1 \cup \dots \cup P_s$ and $f(x)$ can be expressed by $\frac{1}{2}x^\top Q_i x + c_i^\top x + \gamma_i$ on $P_i, i = 1, \dots, s$. Since $F(x)$ is convex and piecewise quadratic, every Q_i must be positive semidefinite. Thus, there exists $\bar{\lambda} > 0$ such that

$$0 \leq \frac{y^\top Q_i y}{y^\top y} \leq \bar{\lambda}$$

for any $y \neq 0$ and all $i = 1, \dots, s$. Then

$$0 \leq \frac{y^\top V y}{y^\top y} \leq \bar{\lambda}, \quad (4)$$

where V is any convex combination of $Q_i, i = 1, \dots, s$.

By a result in nonsmooth analysis [10, Proposition 2.1], $\forall x \in \mathcal{R}^n, \exists V$ so that

$$F(x) = F(x^k) + (g^k)^\top (x - x^k) + \frac{1}{2}(x - x^k)^\top V (x - x^k), \quad (5)$$

where $g^k = \nabla F(x^k)$ and V is a certain convex combination of $Q_i, i = 1, \dots, s$. Define a quadratic function

$$F^+(x) = F(x^k) + (g^k)^\top (x - x^k) + \frac{\bar{\lambda}}{2}(x - x^k)^\top (x - x^k). \quad (6)$$

It is easy to see from (4) and (5) that $F^+(x) \geq F(x)$, $\forall x \in \mathcal{R}^n$. Therefore, we have $F^+(\bar{x}) \geq F(x^{k+1})$, where \bar{x} is the minimum of F^+ along the half-line $x^k + \alpha d^k$ and $x^{k+1} = x^k + \alpha_k d^k$. Notice that since the line search is exact, we have $d^{k-1 \top} g^k = 0$. Thus, if $d^k = 0$ then by (1) we get $g^k = 0$, which implies the optimality of x^k . Therefore in the following we assume $d^k \neq 0$. Notice that because

$$\bar{x} = x^k - \left[\frac{(g^k)^\top d^k}{\lambda(d^k)^\top d^k} \right] d^k, \quad (7)$$

we have

$$\begin{aligned} F(x^{k+1}) - F(x^k) &\leq F^+(\bar{x}) - F(x^k) \\ &= -\frac{[(g^k)^\top d^k]^2}{2\lambda(d^k)^\top d^k}. \end{aligned} \quad (8)$$

From $(d^{k-1})^\top g^k = 0$ and (1) we obtain

$$(g^k)^\top d^k = -\|g^k\|^2 + \beta_k (d^{k-1})^\top g^k = -\|g^k\|^2 \quad (9)$$

and

$$\|d^k\|^2 = \|g^k\|^2 + \beta_k^2 \|d^{k-1}\|^2. \quad (10)$$

It follows from (3) that (10) can be simplified to

$$\frac{\|d_k\|^2}{\|g_k\|^4} = \frac{\|d_{k-1}\|^2}{\|g_{k-1}\|^4} + \frac{1}{\|g_k\|^2}. \quad (11)$$

If $\liminf \nabla F(x^k) \neq 0$, then $\|g^k\| > \gamma > 0$ for some γ and large k . By (11), we get

$$\frac{\|d_k\|^2}{\|g_k\|^4} = \sum_{t=1}^k \frac{1}{\|g_t\|^2} \leq \frac{k}{\gamma^2}. \quad (12)$$

Now (8), (9) and (12) imply that for sufficiently large k

$$F(x^{k+1}) - F(x^k) \leq -\frac{\gamma^2}{2k\lambda}. \quad (13)$$

Thus $\lim F(x^k) = -\infty$. The proof is finished. \square

Corollary 2.3 *If the sequence $x^k \rightarrow x^*$, then x^* is a global minimum of F .*

Proof. In this case $F(x^k) \neq -\infty$ and $\nabla F(x^*) = \liminf \nabla F(x^k) = 0$. \square

3 The reduced conjugate gradient method for (NQP)

3.1 The conjugate gradient method in the reduced space

By substituting $x_B = Rx_N + b$ for x_B in the objective function of (PNQP), the problem further becomes an unconstrained problem for x_N , namely $\min r(x_N) = p(x_B, x_N) = p(Rx_N + b, x_N)$. We apply the conjugate gradient method to the reduced function $r(x_N)$. Note that if $p(x)$ is continuously differentiable, convex, and piecewise quadratic, then so is $r(x_N)$. Suppose that $p(x) = \frac{1}{2}x^\top Qx + c^\top x + \gamma$ at point x for certain Q, c and γ . Let

$$Q = \begin{pmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} c_B \\ c_N \end{pmatrix}.$$

Our algorithm will proceed as (1)-(2) with respect to function $r(x_N)$, i.e.,

$$d_N^k = \begin{cases} -g_N^k, & \text{for } k = 1, \\ -g_N^k + \beta_k d_N^{k-1}, & \text{for } k > 1, \end{cases} \quad (14)$$

and

$$x_N^{k+1} = x_N^k + \alpha_k d_N^k. \quad (15)$$

Since $r(x_N) = p(Rx_N + b, x_N)$, one has

$$g_N^k = \nabla r(x_N^k) = R^\top (Q_{BB}x_B^k + 2Q_{BN}x_N^k + c_B) + 2Q_{NB}x_B^k + Q_{NN}x_N^k + c_N. \quad (16)$$

The computation of (16) can be executed efficiently through the data structure of the tree representation. We shall discuss more details of it in later sections.

Algorithm 3.1 A conjugate gradient method for network convex quadratic programming

Step 0. (Initialization) Arbitrarily choose x^1 such that $x_B^1 = Rx_N^1 + b$ and $Ex^1 = s$ (This includes the process of building the data structures for R), select a starting penalty parameter θ , and set the user-assigned optimality tolerances $\delta > 0$ and $\varepsilon > 0$. Set $k = 1$.

Step 1. (The Conjugate Gradient Iterates) For the given x^k , revise the diagonal of matrix Q and the vector c as follows:

$$\begin{cases} q_{jj} \leftarrow q_{jj}, & \text{if } l_j \leq x_j^k \leq u_j, \\ q_{jj} \leftarrow q_{jj} + \theta, & \text{otherwise,} \end{cases}$$

and

$$\begin{cases} c_j \leftarrow c_j, & \text{if } l_j \leq x_j^k \leq u_j, \\ c_j \leftarrow c_j - 2\theta l_j, & \text{if } x_j^k < l_j, \\ c_j \leftarrow c_j - 2\theta u_j, & \text{if } x_j^k > u_j, \end{cases}$$

where θ is the penalty parameter. Compute the conjugate direction d_N^k of $r(x_N)$ according to (14) and (16). If $g_N^k = 0$, go to Step 2; otherwise, do a line search for $p(x)$ along $d^k = (Rd_N^k, d_N^k)$ to find x^{k+1} , update k , and repeat Step 1.

Step 2. (Optimality Test for (NQP)) Update k . Check whether the new iterate is a δ -feasible and ε -optimal solution of (NQP). If yes, terminate; if no, set $\theta = 10\theta$ and go to Step 1. (The ε -optimality can be checked in the same way as the network simplex method, i.e., computing the duality gaps.)

Note that since $r(x_N) = p(x)$, the line search for $r(x_N + \alpha d_N)$ can be equivalently done by using $p(x_B + \alpha d_B, x_N + \alpha d_N) =: p(x + \alpha d)$. Since $p(x + \alpha d)$ is one-dimensional piecewise quadratic, the line search process is quite simple. Notice that the derivative of $p(x + \alpha d)$ is piecewise linear, we can do the line search by finding the root of $p'_\alpha(x + \alpha d)$. We find that the secant method is a good choice for this job in our numerical test.

3.2 Major implementation schemes

We consider problems of size being $m = 10^2 \sim 10^4$ and $n = 10^4 \sim 10^5$. For this size the storage of matrix $R = -E_B^{-1}E_N$ is difficult if possible at all. Therefore any operations involving R should be done through an efficient data structure. Specifically, we discuss how the two major operations, Rv and $R^\top u$ can be done through the spanning tree B .

It is well known that given a supply vector b satisfying the conservation condition $\sum_i b_i = 0$, it is easy to find x_B such that $E_B x_B = b$. In algebraic language, the columns of E_B can be re-arranged so that E_B is triangle with at most two nonzero entries in each column, so a simple back-substitution process can solve this equation in $O(m)$ time. Working with the spanning tree B , this back substitution corresponds to a

“leaf-folding” algorithm, i.e., select a node of degree one (i.e. a leaf) and compute x_i whose arc is connected to this node, then delete this node and pass its supply to the other node of x_i . Repeat this until all x_i is calculated. A simple implementation of this operation is to record the first arc on the path of a node to the root of the tree (call it **EDGE**(i) for $i \in \mathcal{N}$ - the node set) and the depth of a node (call it **DEPTH**(i) for $i \in \mathcal{N}$), which is the length of the path defining **EDGE**(i). Then the above computation starts from the nodes with the largest depth, using **EDGE** to find the connecting arcs, and repeat for the second deepest nodes, and so on.

Now it is easy to find x_B for any given x_N , since we only have to solve the equation $E_B x_B = b - E_N x_N$. The computation of $b - E_N x_N$ is simple: we just add all fluxes on nonbasic arcs to the supplies at the nodes. Then by employing the leaf-folding algorithm, we obtain x_B . This procedure furnishes the computation of $d_B = R d_N$ as well — we just set $b = 0$ and $x_N = d_N$ in the above.

We next explain how to compute $R^\top u$ for a given u . First we solve $E_B^\top p = -u$ again by using the triangle property of E_B . We interpret u as “voltage” on the basic arcs, then p is the “potential” of the nodes. This time we start from the root and work toward deeper nodes in the tree, setting $p_{ie} - p_{ib} = u_i$ where one of the p_{ib} and p_{ie} is the potential at the calculated nodes and another is the potential at the to-be-calculated node whereas ib is the beginning node of the basic arc i and ie is the ending node of it.

After we find the vector p satisfying $E_B^\top p = -u$, we can easily calculate $w = -E_N^\top p$ by the simple subtraction $w_i = p_{ei} - p_{bi}$ for each nonbasic arc. This is due to the structure of matrix E_N - each of whose column contains exactly one +1 and one -1. Note that we have

$$\begin{pmatrix} u \\ w \end{pmatrix} = - \begin{pmatrix} E_B^\top p \\ E_N^\top p \end{pmatrix} \Rightarrow w = -E_N^\top E_B^{-\top} u \Rightarrow w = R^\top u.$$

The computation needs a list of the two nodes of each arc. Hence we define **BEGIN**(j) and **END**(j) for each $j \in \mathcal{A}$ (the arc set). Note that we need to compute g_N at each step (16) that will require to compute w above. We also compute $d_B = R d_N$, so we compute Rv and $R^\top u$ exactly once at each step for a given u and a given v . All of the computation has to use the tree B . Specifically, use node-length vector **EDGE** and **DEPTH** and arc-length vector **BEGIN** and **END**.

Finally we consider a variety of the problem, in which the objective function $f(x)$ itself is convex piecewise quadratic and C^1 . This case may happen in the stochastic programming problem mentioned in Section 1. Theorems 2.1–2.2 are all valid for the new case with little changes in the proofs. On the algorithmic aspects, once Q and c are defined, the same procedure as Step 1 can be used for finding the conjugate directions. The line search procedure remains unchanged, only more pieces of the function might be checked in order to find the minimum.

3.3 Some implementation details

All initial points are set as $x_N^0 = 0$ and $x_B^0 = -E_B^{-1}s$. Note that this kind of initialization does not necessarily meet the simple bounds $l \leq x \leq u$ even if $l < 0 < u$. The initial penalty parameter is set to be 10.

We use the following criteria to stop updating the penalty parameter. Let x_θ^k be the minimum point of penalty function in the k -th iteration, check

$$|p(x_\theta^k) - p(x_\theta^{k-1})| \leq \varepsilon \max(|p(x_\theta^k)|, |p(x_\theta^{k-1})|)$$

for the less progress of the penalty function. We use the relative bounds for δ -feasibility:

$$\frac{1}{2} \sum_{j=1}^n [\min^2\{x_j - l_j, 0\} + \min^2\{u_j - x_j, 0\}] \leq \delta \|u - l\|^2. \quad (17)$$

If both x_θ^k and x_θ^{k-1} are δ -feasible, and we make less progress of the penalty function, then we stop the algorithm. Here the parameters δ, ε are set to be 10^{-4} and 10^{-7} respectively.

For the inner iteration of conjugate gradient method, the stopping criteria are

$$|p(x^k) - p(x^{k-1})| \leq \varepsilon \max(|p(x^k)|, |p(x^{k-1})|)$$

or

$$\|g_N^k\| \leq \varepsilon,$$

where g_N^k is the reduced gradient of the penalty function, x^k is the k -th iterate in the conjugate gradient method. Note that $\{x_\theta^k\}$ is a subsequence of $\{x^k\}$, namely that x_θ^k is the optimal solution of the conjugate gradient method with parameter θ , which in our implementation is $10^p, p = 1, 2, \dots$

The stopping criteria of the line search are

$$|p'_\alpha(x^k + \alpha d^k)|_{\alpha=\alpha_k} \leq \bar{\varepsilon} |p'_\alpha(x^k + \alpha d^k)|_{\alpha=0}$$

or the line search step is fixed in an interval with its length less than $\bar{\varepsilon}/\theta$. Such an interval is given in the process of the secant method with its length decreasing. Here to have a more exact line search, we let $\bar{\varepsilon} = 10^{-9}$.

4 Numerical results

4.1 Generate random network problems

We test convex separable network problems. The reasons to test separable problems are twofold. 1. They are most common in practice; 2. We would like to know the essential behavior of conjugate gradient method rather than numerical skills of handling non-separable data. An improper treatment of a great deal of non-separable data would certainly make the conjugate gradient method look bad even if the true reason is not due to the method itself.

We test two kinds of problems. The main difference between them is on their network topologies: one is the complete graph, and the other is not and its arcs are generated randomly with a prespecified number of arcs. For simplicity, we call these two types of topologies “full” and “sparse”, respectively.

Given the number of node m , we list all $m(m-1)/2$ arcs (ignoring their directions) by a natural order so that the first $m-1$ arcs form a spanning tree:

$$(1, 2), (2, 3), \dots, (m-1, m), (1, 3), (2, 4), \dots, (m-2, m), \dots, (1, m).$$

For the sparse network of n arcs, we generate random integers $\xi_1 < \xi_2 < \dots < \xi_{n-m+1}$ with all $\xi_j \in [m, m(m-1)/2]$. Pick the all the ξ_j -th arcs from the list above mentioned and randomly set their directions, we have the sparse network with all arcs in spanning tree remaining unchanged.

The other data in (NQP) are generated as follows. The entries of diagonal matrix Q are generated in the interval $[1, n]$, which forms a diagonal matrix with condition number $O(n)$. The vectors c, l, u are set freely with $l < u$. To make sure that the problem we generate is feasible, we let $s = E(l+u)/2$, i.e., we have a feasible solution $x = (l+u)/2$.

4.2 Performance of our algorithms

The test results are given in Table 4.1. The words “full” and “spar” with the latter for “sparse” are self-explanatory. The maximum number of arc is up to 40,000. In Table 4.1, NF and NG stand for the numbers of function and gradient evaluations, respectively; CPU is the total running time in seconds including the time for problem generation; θ^* is the last penalty parameter; $f(x^0)$ and $f(x^*)$ are the initial and optimal objective values, respectively. The source codes are written in Fortran and C and are running in an HP workstation.

#	type	nodes	arcs	NF	NG	CPU(s)	θ^*	$f(x^0)$	$f(x^*)$
1	full	40	780	1210	6317	1.14	1.0e09	1.60e08	1.96e05
2	full	80	3160	3793	22685	16.14	1.0e08	1.11e10	6.53e06
3	full	120	7140	6026	29643	54.36	1.0e08	3.98e11	4.58e07
4	full	160	12720	8204	40184	145.00	1.0e07	1.38e12	2.01e08
5	full	200	19900	11196	52580	284.39	1.0e07	1.12e13	5.94e08
6	full	240	28680	14078	65928	520.92	1.0e07	9.04e13	1.71e09
7	full	280	39060	18571	89415	1004.42	1.0e07	5.77e13	3.02e09
8	spar	1000	10000	12722	62087	161.83	1.0e09	1.38e14	5.82e10
9	spar	2000	15000	15187	79433	309.31	1.0e09	1.00e16	7.43e11
10	spar	3000	20000	20381	104782	549.83	1.0e10	3.32e16	3.28e12
11	spar	4000	25000	18702	99992	660.92	1.0e09	7.40e16	1.04e13
12	spar	5000	30000	24188	127901	1024.83	1.0e09	1.45e17	2.36e13
13	spar	6000	35000	24987	132682	1245.77	1.0e09	2.87e17	5.00e13
14	spar	7000	40000	29585	152322	1694.02	1.0e09	1.05e18	8.79e13

Table 4.1: Performances of 14 random (NQP) problems

4.3 Extension to multicommodity problems

We consider the following multicommodity quadratic network problem:

$$\text{minimize } f(x) = \sum_{i=1}^{n_c} \frac{1}{2} x_i^\top Q_i x_i + c_i^\top x_i \quad \text{subject to } Ex_i = s_i, \quad l_i \leq x_i \leq u_i, \quad \forall i = 1, 2, \dots, n_c, \quad (\text{MCNQP})$$

where n_c is the number of commodities. The other notations remain unchanged, e.g., for each i , Q_i is a positive diagonal matrix, l_i, u_i are the simple bounds and s_i is the supply vector for commodity i . The network topology of our multicommodity problem is sparse. We stress two points. 1. The number of variables for multicommodity problem is $n \cdot n_c$, where n is the number of arcs. 2. All the commodities share the same network topology E which can be exploited so that the solution time of (MCNQP) is much faster than the solution time of an (NQP) of size $n \cdot n_c$.

It should be noted that a practical multicommodity network flow model often has additional linear inequality constraints such as the total flow capacity constraints on the arcs. Under the current algorithmic framework, those constraints can be incorporated into the objective function as additional quadratic penalty terms, therefore making no essential difference from the model considered here.

In order to implement our algorithm on the multicommodity problem, all the settings for the problem and algorithm in subsection 3.3 and 4.1 remain unchanged, except that the matrices Q_i and vectors c_i are modified. In our test, each entries of Q_i is randomly generated in the interval $[1, n \cdot n_c]$, and c_i are scaled to have larger components comparing those in problem (NQP).

The numerical results are reported in Table 4.2, where n_c stands for the number of commodities, and other notations share the same meanings to Table 4.1. Note that in Table 4.2, even the smallest problem has more variables than any problem in Table 4.1. The largest problem in the table has a thousand arcs and a hundred commodities which amounts to 10^5 variables.

5 Conclusions

This paper combines a well-know method (the conjugate gradient method) for nonlinear programming and a well-know data structure (the spanning tree structure) of the network flow problem to develop a solu-

#	n_c	nodes	arcs	NF	NG	CPU(s)	θ^*	$f(x^0)$	$f(x^*)$
1	70	50	650	4163	20959	1330.11	1.0e11	7.70e11	2.11e10
2	70	100	700	3619	18672	1285.11	1.0e11	5.14e12	2.13e10
3	80	150	750	3530	18374	1556.20	1.0e11	2.17e13	1.12e11
4	80	200	800	3441	18220	1644.62	1.0e11	5.54e13	2.88e11
5	90	250	850	3436	17979	1971.78	1.0e11	1.50e14	8.14e11
6	90	300	900	3497	18453	2143.76	1.0e11	2.54e14	1.52e12
7	100	350	950	3535	18722	2564.09	1.0e11	5.98e14	3.23e12
8	100	400	1000	3447	18162	2604.53	1.0e11	7.42e14	5.19e12

Table 4.2: Performances of 8 random multicommodity network problems

tion technique for minimization of convex (probably non-separable) quadratic network programs. By using quadratic penalty functions, we preserve the smoothness of the objective function and change it to a convex piecewise quadratic function. We show that the Fletcher-Reeves method is globally convergent in this case and report our numerical results that show the efficiency of the algorithm.

It would be interesting to extend our algorithm to nonconvex network quadratic optimization problems. To achieve this goal, we need to develop results similar to Theorem 2.1 and Theorem 2.2. It appears that in this case the conjugate gradient method would still converge to a stationary point in the sense of $\liminf \|\nabla p(x_k)\| = 0$. There are many studies on the general convergence properties of conjugate gradient methods (e.g., [1, 5, 6, 11]). Some of them require the objective function to be twice continuously differentiable and thus the results are not applicable to our case. Of particular interest to us would be the conjugate gradient methods that are convergent when applied to functions with only Lipschitz continuous gradient, of which the function $p(\cdot)$ in this paper is a special case.

References

- [1] Y.H. Dai and Y. Yuan, A nonlinear conjugate gradient method with a strong global convergence property, *SIAM J. Optimization*, **10** (1999) 177–182.
- [2] R. Fletcher, *Practical Method of Optimization, Vol I: Unconstrained Optimization*, 2nd edition, Wiley, 1987.
- [3] A.B. Gamble, A.R. Conn and W.R. Pulleyblank, A network penalty problem, *Mathematical Programming*, **50** (1991) 53–73.
- [4] P. Kall and S.W. Wallace, *Stochastic Programming*, Wiley, 1994.
- [5] B. Polak and G. Ribière, Note sur la convergence des méthodes de directions conjuguées, *Rev. Fran. Informat. Rech. Opér.*, **16** (1969) 35–43.
- [6] M.J.D. Powell, Nonconvex minimization calculations and the conjugate gradient method, *Lecture Notes in Mathematics*, **1066** (1984) 121–141.
- [7] R.T. Rockafellar, *Convex Analysis*, Princeton University Press, 1970.
- [8] R.T. Rockafellar, *Network Flows and Monotropic Optimization*, Wiley, 1984.
- [9] J. Sun and H. Kuo, Applying a Newton method to strictly convex separable network quadratic programs, *SIAM J. Optimization*, **8** (1998) 728–745.
- [10] J. Sun and L. Qi, “An Interior point algorithm of $O(\sqrt{m}|\ln \epsilon|)$ iterations for C^1 -convex programming”, *Mathematical Programming*, **57** (1992) 239–257.
- [11] J. Sun and J. Zhang, “Global convergence of conjugate gradient methods without line search”, *Annals of Operations Research*, 103 (2001) 161–173.