versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

FORTRAN SUBROUTINES FOR NETWORK FLOW OPTIMIZATION USING AN INTERIOR POINT ALGORITHM

L. F. Portugal
Dep. de Ciências da Terra – Universidade de Coimbra
Coimbra – Portugal

M. G. C. Resende*
Internet and Network Systems Research Center
AT&T Labs Research
Florham Park – NJ, USA
mgcr@research.att.com

G. Veiga
HTF Software
Rio de Janeiro – RJ, Brazil
gveiga@gmail.com

J. Patrício
Instituto Politécnico de Tomar – Tomar – Portugal e
Instituto de Telecomunicações – Polo de Coimbra – Portugal
Joao.Patricio@aim.estt.ipt.pt

J. J. Júdice
Dep. de Matemática – Univ. de Coimbra – Coimbra e
Instituto de Telecomunicações – Polo de Coimbra – Portugal
Joaquim.Judice@co.it.pt

* *Corresponding author* / autor para quem as correspondências devem ser encaminhadas

Recebido em 01/2007; aceito em 04/2008
Received January 2007; accepted April 2008

Abstract

We describe Fortran subroutines for network flow optimization using an interior point network flow algorithm, that, together with a Fortran language driver, make up PDNET. The algorithm is described in detail and its implementation is outlined. Usage of the package is described and some computational experiments are reported. Source code for the software can be downloaded at `http://www.research.att.com/~mgcr/pdnet`.

Keywords: optimization; network flow problems; interior point method; conjugate gradient method; FORTRAN subroutines.

Resumo

É apresentado o sistema PDNET, um conjunto de subrotinas em Fortran para a otimização de fluxos lineares em redes utilizando um algoritmo de pontos interiores. O algoritmo e a sua implementação são descritos com algum detalhe. A utilização do sistema é explicada e são apresentados alguns resultados computacionais. O código fonte está disponível em `http://www.research.att.com/~mgcr/pdnet`.

Palavras-chave: otimização; problemas de fluxo em rede; método de ponto interior; método do gradiente conjugado; subrotinas FORTRAN.

1. Introduction

Given a directed graph $G=(\mathcal{V},\mathcal{E})$, where $\mathcal{V}$ is a set of m vertices and $\mathcal{E}$ a set of n edges, let (i,j) denote a directed edge from vertex i to vertex j. The minimum cost network flow problem can be formulated as

$$\min \sum_{(i,j)\in\mathcal{E}} c_{ij}x_{ij}$$

subject to

$$\sum_{(i,j)\in\mathcal{E}} x_{ij} - \sum_{(j,i)\in\mathcal{E}} x_{ji} = b_i,\ i\in\mathcal{V} \tag{1}$$

$$l_{ij} \le x_{ij} \le u_{ij},\ (i,j)\in\mathcal{E}, \tag{2}$$

where x_{ij} denotes the flow on edge (i,j) and c_{ij} is the cost of passing one unit of flow on edge (i,j). For each vertex $i\in\mathcal{V}$, b_i denotes the flow produced or consumed at vertex i. If $b_i>0$, vertex b_i is a source. If $b_i<0$, vertex i is a sink. Otherwise ($b_i=0$), vertex i is a transshipment vertex.

For each edge $(i,j)\in\mathcal{E}$, l_{ij} (u_{ij}) denotes the lower (upper) bound on flow on edge (i,j). Most often, the problem data are assumed to be integer. In matrix notation, the above network flow problem can be formulated as a primal linear program of the form

$$\min\{c^\top x \mid Ax=b; x+s=u; x,s\ge 0\} \tag{3}$$

where c is a $m\times n$ vector whose elements are c_{ij}, A is the $m\times n$ *vertex-edge incidence matrix* of the graph $G=(\mathcal{V},\mathcal{E})$, i.e. for each edge (i,j) in $\mathcal{E}$ there is an associated column in matrix A with exactly two nonzero entries: an entry 1 in row i and an entry -1 in row j; b, x, and u are defined as above, and s is an n-dimensional vector of upper bound slacks. Furthermore, an appropriate variable change allows us to assume that the lower bounds l are zero, without loss of generality. The dual of (3) is

$$\max\{b^\top y - u^\top w \mid A^\top y - w + z = c; z,w\ge 0\} \tag{4}$$

where y is the m-dimensional vector of dual variables and w and z are n-dimensional vectors of dual slacks.

If graph G is disconnected and has p connected components, there are exactly p redundant flow conservation constraints, which are sometimes removed from the problem formulation. Without loss of generality, we rule out trivially infeasible problems by assuming

$$\sum_{j\in\mathcal{V}^k} b_j = 0,\ k=1,\ldots,p,$$

where $\mathcal{V}^k$ is the set of vertices for the k-th component of G.

If it is required that the flow x_{ij} be integer, (2) is replaced with

$$l_{ij} \le x_{ij} \le u_{ij}, x_{ij} \text{ integer}, (i,j)\in\mathcal{E}.$$

In the description of the algorithm, we assume without loss of generality, that $l_{ij} = 0$ for all $(i, j) \in \mathcal{E}$ and that $c \neq 0$. A simple change of variables is done in the subroutines to transform the original problem into an equivalent one with $l_{ij} = 0$ for all $(i, j) \in \mathcal{E}$. The flow is transformed back to the original problem upon termination. The case where $c = 0$ is a simple feasibility problem, and is handled by solving a maximum flow problem [1].

Before concluding this introduction, we present some notation and outline the remainder of the paper. We denote the i-th column of A by A_i, the i-th row of A by $A_{i\cdot}$ and a submatrix of A formed by columns with indices in set S by A_S. Let $x \in \mathbb{R}^n$. We denote by X the $n \times n$ diagonal matrix having the elements of x in the diagonal. The Euclidean or 2-norm is denoted by $\|\cdot\|$.

This paper describes Fortran subroutines used in an implementation of PDNET, an interior point network flow method introduced in Portugal, Resende, Veiga & Júdice [12]. The paper is organized as follows. In Section 2 we review the truncated infeasible-primal feasible-dual interior point method for linear programming. The implementation of this algorithm to handle network flow problems is described in Section 3. Section 4 describes the subroutines and their usage. Computational results, comparing PDNET with the network optimizer in CPLEX 10, are reported in Section 5. Concluding remarks are made in Section 6.

2. Truncated primal-infeasible dual-feasible interior point algorithm

In this section, we recall the interior point algorithm implemented in PDNET. Let

$$\mathcal{Q}_+ = \{(x, y, s, w, z) \in \mathbb{R}^{n+m+n+n+n} : x > 0, s > 0, w > 0, z > 0\}$$

$$\mathcal{S}_+ = \{(x, y, s, w, z) \in \mathcal{Q}_+ : x + s = u,\ A^\top y - w + z = c\}\ .$$

The truncated primal-infeasible dual-feasible interior point (TPIDF) algorithm [12] starts with any solution $(x^0, y^0, s^0, w^0, z^0) \in \mathcal{S}_+$. At iteration k, the Newton direction $(\Delta x^k, \Delta y^k, \Delta s^k, \Delta w^k, \Delta z^k)$ is obtained as the solution of the linear system of equations

$$\begin{cases} A\Delta x^k &= b - Ax^k + r^k, \\ \Delta x^k + \Delta s^k &= 0, \\ A^\top \Delta y^k - \Delta w^k + \Delta z^k &= 0, \\ Z^k \Delta x^k + X^k \Delta z^k &= \mu_k e - X^k Z^k e, \\ W^k \Delta s^k + S^k \Delta w^k &= \mu_k e - W^k S^k e, \end{cases} \tag{5}$$

where e is a vector of ones of appropriate order, r^k is such that

$$\| r^k \| \leq \beta_0 \| Ax^k - b \|, 0 \leq \beta_0 < \beta_1, \tag{6}$$

with $\beta_1 = 0.1$, and

$$\mu_k = \beta_1 \frac{(x^k)^\top z^k + (w^k)^\top s^k}{2n}\ . \tag{7}$$

Primal and dual steps are taken in the direction $(\Delta x^k, \Delta y^k, \Delta s^k, \Delta w^k, \Delta z^k)$ to compute new iterates according to

$$\begin{aligned} x^{k+1} &= x^k + \alpha_p \Delta x^k, \\ s^{k+1} &= s^k + \alpha_p \Delta s^k, \\ y^{k+1} &= y^k + \alpha_d \Delta y^k, \\ w^{k+1} &= w^k + \alpha_d \Delta w^k, \\ z^{k+1} &= z^k + \alpha_d \Delta z^k, \end{aligned} \tag{8}$$

where α_p and α_d are step-sizes in the primal and dual spaces, respectively, and are given by

$$\begin{aligned} \alpha_p &= \varrho_p \max\{\alpha : x^k + \alpha \Delta x^k \geq 0, s^k + \alpha \Delta s^k \geq 0\}, \\ \alpha_d &= \varrho_d \max\{\alpha : w^k + \alpha \Delta w^k \geq 0, z^k + \alpha \Delta z^k \geq 0\}, \end{aligned} \tag{9}$$

where $\varrho_p = \varrho_d = 0.9995$.

The solution of the linear system (5) is obtained in two steps. First, we compute the Δy^k component of the direction as the solution of the system of normal equations

$$A\Theta^k A^\top \Delta y^k = -A\Theta^k(\mu_k (X^k)^{-1} e - \mu_k (S^k)^{-1} e - c + A^\top y^k) + (b - Ax^k), \tag{10}$$

where Θ_k is given by

$$\Theta^k = (Z^k (X^k)^{-1} + W^k (S^k)^{-1})^{-1}. \tag{11}$$

The remaining components of the direction are then recovered by

$$\begin{aligned} \Delta x^k &= \Theta^k A^\top \Delta y^k + \Theta^k(\mu_k (X^k)^{-1} e - \mu_k (S^k)^{-1} e - c + A^\top y^k), \\ \Delta s^k &= -\Delta x^k, \\ \Delta z^k &= -z^k + \mu_k (X^k)^{-1} e - Z^k (X^k)^{-1} \Delta x^k, \\ \Delta w^k &= -w^k + \mu_k (S^k)^{-1} e - W^k (S^k)^{-1} \Delta s^k. \end{aligned} \tag{12}$$

3. Implementation

We discuss how the truncated primal-infeasible dual-feasible algorithm can be used for solving network flow problems. For ease of discussion, we assume, without loss of generality, that the graph G is connected. However, disconnected graphs are handled by PDNET.

3.1 Computing the Newton direction

Since the exact solution of (10) can be computationally expensive, a preconditioned conjugate gradient (PCG) algorithm is used to compute approximately an interior point search direction at each iteration. The PCG solves the linear system

$$M^{-1}(A\Theta^k A^\top)\Delta y^k = M^{-1}\overline{b}, \tag{13}$$

where M is a positive definite matrix and Θ^k is given by (11), and

$$\overline{b} = -A\Theta^k(\mu_k (X^k)^{-1} e - \mu_k (S^k)^{-1} e - c + A^\top y^k) + (b - Ax^k).$$

The aim is to make the preconditioned matrix

$$M^{-1}(A\Theta^k A^\top) \tag{14}$$

less ill-conditioned than $A\Theta^k A^\top$, and improve the efficiency of the conjugate gradient algorithm by reducing the number of iterations it takes to find a feasible direction.

Pseudo-code for the preconditioned conjugate gradient algorithm implemented in PDNET is presented in Figure 1. The matrix-vector multiplications in line 7 are of the form $A\Theta^k A^\top p_i$, and can be carried out without forming $A\Theta^k A^\top$ explicitly. PDNET uses as its initial direction Δy_0 the direction Δy produced in the previous call to the conjugate gradient algorithm, i.e. during the previous interior point iteration. The first time `pcg` is called, we assume $\Delta y_0 = (0,\ldots,0)$.

The preconditioned residual is computed in lines 3 and 11 when the system of linear equations

$$Mz_{i+1} = r_{i+1}, \tag{15}$$

is solved. PDNET uses primal-dual variants of the diagonal and spanning tree preconditioners described in [15,16].

Procedure `pcg` ($A, \Theta^k, \bar{b}, \epsilon_{cg}, \Delta y$)

1. $\Delta y_0 := \Delta y$;
2. $r_0 := \bar{b} - A\Theta^k A^\top \Delta y_0$;
3. $z_0 := M^{-1} r_0$;
4. $p_0 := z_0$;
5. $i := 0$;
6. **Do** stopping criterion not satisfied $\rightarrow$
7. $\quad q_i := A\Theta^k A^\top p_i$;
8. $\quad \alpha_i := z_i^\top r_i / p_i^\top q_i$;
9. $\quad \Delta y_{i+1} := \Delta y_i + \alpha_i p_i$;
10. $\quad r_{i+1} := r_i - \alpha_i q_i$;
11. $\quad z_{i+1} := M^{-1} r_{i+1}$;
12. $\quad \beta_i := z_{i+1}^\top r_{i+1} / z_i^\top r_i$;
13. $\quad p_{i+1} := z_{i+1} + \beta_i p_i$;
14. $\quad i := i + 1$;
15. **Od**;
16. $\Delta y := \Delta y_i$

End pcg

Figure 1 – The preconditioned conjugate gradient algorithm.

The diagonal preconditioner, $M = \text{diag}(A\Theta^k A^\top)$, can be constructed in $O(n)$ operations, and makes the computation of the preconditioned residual of the conjugate gradient possible with $O(m)$ divisions. This preconditioner has been shown to be effective during the initial interior point iterations [11,14,15,16,18].

In the spanning tree preconditioner [16], one identifies a maximal spanning tree of the graph G, using as weights the diagonal elements of the current scaling matrix,

$$w = \Theta^k e,$$

where e is a unit n-vector. An exact maximal spanning tree is computed with the Fibonacci heap variant of Prim's algorithm [13], as described in [1]. At the k-th interior point iteration, let $\mathcal{T}^k = \{t_1,\ldots,t_q\}$ be the indices of the edges of the maximal spanning tree. The spanning tree preconditioner is

$$M = A_{\mathcal{T}^k}\Theta^k_{\mathcal{T}^k}A^\top_{\mathcal{T}^k},$$

where

$$\Theta^k_{\mathcal{T}^k} = \text{diag}(\Theta^k_{t_1},\ldots,\Theta^k_{t_q}).$$

For simplicity of notation, we include in $A_{\mathcal{T}^k}$ the linear dependent rows corresponding to the redundant flow conservation constraints. At each conjugate gradient iteration, the preconditioned residual system

$$Mz_{i+1} = r_{i+1}$$

is solved with the variables corresponding to redundant constraints set to zero. Since $A_{\mathcal{T}^k}$ can be ordered into a block diagonal form with triangular diagonal blocks, then the preconditioned residuals can be computed in $O(m)$ operations.

A heuristic is used to select the preconditioner. The initial selection is the diagonal preconditioner, since it tends to outperform the other preconditioners during the initial interior point iterations. The number of conjugate gradients taken at each interior point iteration is monitored. If the number of conjugate gradient iterations exceeds $\sqrt{m}/4$, the current computation of the direction is discarded, and a new conjugate gradient computation is done with the spanning tree preconditioner. The diagonal preconditioner is not used again. The diagonal preconditioner is limited to at most 30 interior point iterations. If at iteration 30 the diagonal preconditioner is still in effect, at iteration 31 the spanning tree preconditioner is triggered. Also, as a safeguard, a hard limit of 1000 conjugate gradient iterations is imposed.

To determine when the approximate direction Δy_i produced by the conjugate gradient algorithm is satisfactory, one can compute the angle θ between

$$(A\Theta^k A^\top)\Delta y_i \text{ and } \bar{b}$$

and stop when $|1-\cos\theta| < \epsilon^k_{cos}$, where ϵ^k_{cos} is the tolerance at interior point iteration k [8,15]. PDNET initially uses $\epsilon^0_{cos} = 10^{-3}$ and tightens the tolerance after each interior point iteration k as follows:

$$\epsilon^{k+1}_{cos} = \epsilon^k_{cos} \times \Delta\epsilon_{cos},$$

where, in PDNET, $\Delta\epsilon_{cos} = 0.95$. The exact computation of

$$\cos\theta = \frac{|\bar{b}^\top (A\Theta^k A^\top)\Delta y_i\ |}{\|\bar{b}\|\cdot\|(A\Theta^k A^\top)\Delta y_i\|}$$

has the complexity of one conjugate gradient iteration and should not be carried out every conjugate gradient iteration. Since $(A\Theta^k A^\top)\Delta y_i$ is approximately equal to $\bar{b} - r_i$, where r_i is the estimate of the residual at the i-th conjugate gradient iteration, then

$$\cos\theta \approx \frac{|\bar{b}^\top (\bar{b} - r_i)|}{\|\bar{b}\|\cdot\|(\bar{b} - r_i)\|} .$$

Since, on network linear programs, the preconditioned conjugate gradient method finds good directions in few iterations, this estimate is quite accurate in practice. Since it is inexpensive, it is computed at each conjugate gradient iteration.

3.2 Stopping criteria for interior point method

In [15], two stopping criteria for the interior point method were used. The first, called the primal-basic (PB) stopping rule, uses the spanning tree computed for the tree preconditioner. If the network flow problem has a unique solution, the edges of the tree converge to the optimal basic sequence of the problem. Let $\mathcal{T}$ be the index set of the edges of the tree, and define

$$\Omega^+ = \{i \in \{1,2,\ldots,n\} \setminus \mathcal{T} : x_i / z_i > s_i / w_i\}$$

to the index set of edges that are fixed to their upper bounds. If the solution $x_{\mathcal{T}}^*$ of the linear system

$$A_{\mathcal{T}} x_{\mathcal{T}}^* = b - \sum_{i\in\Omega^+} u_i A_i ,$$

is such that $0 \le x_{\mathcal{T}}^* \le u$, then $x_{\mathcal{T}}^*$ is a feasible basic solution. Furthermore, if the data is integer, then $x_{\mathcal{T}}^*$ has only integer components. Optimality of $x_{\mathcal{T}}^*$ can be verified by computing a lower bound on the optimal objective function value. This can be done with a strategy introduced independently in [15] and [10,17]. Denote by x_i^* the i-th component of $x_{\mathcal{T}}^*$ and let

$$\mathcal{F} = \{i \in \mathcal{T} : 0 < x_i^* < u_i\} .$$

A tentative optimal dual solution y^* (having a possibly better objective value than the current dual interior point solution y^k) can be found by orthogonally projecting y^k onto the supporting affine space of the dual face complementary to $x_{\mathcal{T}}^*$. In an attempt to preserve dual feasibility, we compute y^* as the solution of the least squares problem

$$\min_{y^*\in\mathbb{R}^m} \{\| y^* - y^k \| : A_{\mathcal{F}}^\top y^* = c_{\mathcal{F}}\} .$$

Resende & Veiga [15] describe a $O(m)$ operation procedure to compute this projection. A feasible dual solution (y^*, z^*, w^*) is built by adjusting the dual slacks. Let $\delta_i = c_i - A_i^\top y^*$. Then,

$$w_i^* = \begin{cases} -\delta_i & \text{if } \delta_i < 0 \\ 0 & \text{otherwise} \end{cases} \qquad z_i^* = \begin{cases} 0 & \text{if } \delta_i < 0 \\ \delta_i & \text{otherwise} \end{cases}.$$

If $c^\top x^* - b^\top y^* + u^\top w^* = 0$, then (x^*, s^*) and (y^*, w^*, z^*) are optimal primal and dual solutions, respectively. If the data is integer and $0 < c^\top x^* - b^\top y^* + u^\top w^* < 1$, (x^*, s^*) is a primal optimal (integer) solution.

To apply the second stopping procedure of [15], called the maximum flow (MF) stopping criterion, an indicator function to partition the edge set into active and inactive (fixed at upper or lower bounds) is needed. In PDNET, the indicator used is the so-called primal-dual indicator, studied by Gay [5] and El-Bakry, Tapia & Zhang [4]. Let ξ be a small tolerance. Edge i is classified as inactive at its lower bound if

$$\frac{x_i}{z_i} < \xi \text{ and } \frac{s_i}{w_i} > \xi^{-1}.$$

Edge i is classified as inactive at its upper bound if

$$\frac{x_i}{z_i} > \xi^{-1} \text{ and } \frac{s_i}{w_i} < \xi.$$

The remaining edges are set active. In PDNET, ξ is initially set to 10^{-3} and this tolerance is tightened each time the MF test is triggered according to $\xi^{new} = \xi^{old} \times \Delta\xi$, where, in PDNET, $\Delta\xi = 0.95$.

We select a tentative optimal dual face $\mathcal{F}$ as a maximum weighted spanning forest limited to the active edges as determined by the indicator. The edge weights used in PDNET are those of the scaling matrix Θ^k.

As in the PB indicator, we project the current dual interior solution y^k orthogonally onto $\mathcal{F}$. Once the projected dual solution y^* is at hand, we attempt to find a feasible flow x^* complementary to y^*. A refined tentative optimal face is selected by redefining the set of active edges as

$$\widetilde{\mathcal{F}} = \{i \in \{1,2,\ldots,n\} : | c_i - A_i^\top y^* | < \epsilon_r\},$$

where ϵ_r is a small tolerance ($\epsilon_r = 10^{-8}$ in PDNET). The method attempts to build a primal feasible solution, x^*, complementary to the tentative dual optimal solution by setting the inactive edges to lower or upper bounds, i.e., for $i \in \{1,2,\ldots,n\} \setminus \widetilde{\mathcal{F}}$,

$$x_i^* = \begin{cases} 0 & \text{if } i \in \Omega^- = \{j \in \{1,2,\ldots,n\} \setminus \widetilde{\mathcal{F}} : c_j - A_j^\top y^* > 0\} \\ u_i & \text{if } i \in \Omega^+ = \{j \in \{1,2,\ldots,n\} \setminus \widetilde{\mathcal{F}} : c_j - A_j^\top y^* < 0\} \end{cases}.$$

By considering only the active edges, a *restricted network* is built. Flow on this network must satisfy

$$A_{\widetilde{\mathcal{F}}} x_{\widetilde{\mathcal{F}}} = \tilde{b} = b - \sum_{i \in \Omega^+} u_i A_i \,, \tag{16}$$

$$0 \le x_i \le u_i,\ i \in \widetilde{\mathcal{F}} \,.$$

Clearly, from the flow balance constraints (16), if a feasible flow $x^*_{\widetilde{\mathcal{F}}}$ for the restricted network exists, it defines, along with $x^*_{\Omega^+}$ and $x^*_{\Omega^-}$, a primal feasible solution complementary to y^*. A feasible flow for the restricted network can be determined by solving a maximum flow problem on the *augmented network* defined by the underlying graph $\tilde{G} = (\widetilde{\mathcal{V}}, \tilde{\mathcal{E}})$, where

$$\widetilde{\mathcal{V}} = \{\sigma\} \cup \{\pi\} \cup \mathcal{V} \text{ and } \tilde{\mathcal{E}} = \Sigma \cup \Pi \cup \widetilde{\mathcal{F}} \,.$$

In addition, for each edge $(i, j) \in \widetilde{\mathcal{F}}$ there is an associated capacity u_{ij}. Let

$$\mathcal{V}^+ = \{i \in \mathcal{V} : \tilde{b}_i > 0\} \text{ and } \mathcal{V}^- = \{i \in \mathcal{V} : \tilde{b}_i < 0\} \,.$$

The additional edges are such that $\Sigma = \{(\sigma, i) : i \in \mathcal{V}^+\}$, with associated capacity $\tilde{b}_i$ for each edge (σ, i), and $\Pi = \{(i, \pi) : i \in \mathcal{V}^-\}$, with associated capacity $-\tilde{b}_i$ for each edge (i, π). It can be shown that if $\mathcal{M}_{\sigma,\pi}$ is the maximum flow value from σ to π, and $\tilde{x}$ is a maximal flow on the augmented network, then $\mathcal{M}_{\sigma,\pi} = \sum_{i \in \mathcal{V}^+} \tilde{b}_i$ if and only if $\tilde{x}_{\widetilde{\mathcal{F}}}$ is a feasible flow for the restricted network [15]. Therefore, finding a feasible flow for the restricted network involves the solution of a maximum flow problem. Furthermore, if the data is integer, this feasible flow is integer, as we can select a maximum flow algorithm that provides an integer solution.

Since this stopping criterion involves the solution of a maximum flow problem, it should not be triggered until the interior point algorithm is near the optimal solution. The criterion is triggered at iteration k, when $\mu_k < \epsilon_\mu$ occurs for first time. The choice $\epsilon_\mu = 1$ used in PDNET is appropriate for the set of test problems considered here. In a more general purpose implementation, a scale invariant criterion is desirable. All subsequent iterations test this stopping rule. In PDNET, the implementation of Goldfarb & Grigoriadis [6] of Dinic's algorithm is used to solve the maximum flow problems.

3.3 Other implementation issues

To conclude this section, we make some remarks on other important implementation issues of the primal-infeasible, dual-feasible algorithm, namely the starting solution, the adjustment of parameter μ_k, and the primal and dual stepsizes.

Recall that the algorithm starts with any solution $\{x^0, s^0, y^0, w^0, z^0\}$ satisfying

$$x^0 > 0, s^0 > 0, w^0 > 0, z^0 > 0 \,, \tag{17}$$

(18) $$x^0 + s^0 = u \ ,$$

and

(19) $$A^\top y^0 - w^0 + z^0 = c \ ,$$

but does not have to satisfy $Ax^0 = b$. Additionally, it is desirable that the initial point also satisfy the remaining equations that define the central path (5), i.e.

(20) $$X^0 z^0 = \mu e \ \text{ and } \ S^0 w^0 = \mu e \ ,$$

for $\mu > 0$. For $(i, j) \in \mathcal{E}$, let

$$\begin{aligned} x_{ij}^0 &= \nu_{ij} u_{ij}, \\ s_{ij}^0 &= (1-\nu_{ij})\, u_{ij}, \\ z_{ij}^0 &= \mu/(\nu_{ij}\, u_{ij}), \\ w_{ij}^0 &= \mu/((1-\nu_{ij})\, u_{ij}), \end{aligned}$$

be the starting solution, where $0 < \nu_{ij} < 1$ and $\mu > 0$. It is easy to verify that this starting solution satisfies (17-18) as well as (20).

Condition (19) is satisfied if, for $(i, j) \in \mathcal{E}$, ν_{ij} is chosen as

$$\nu_{ij} = \begin{cases} \dfrac{1}{2} + \dfrac{\mu}{\vartheta_{ij} u_{ij}} - \sqrt{\dfrac{1}{4} + \left(\dfrac{\mu}{\vartheta_{ij} u_{ij}}\right)^2} & \text{if } \vartheta_{ij} > 0, \\ \dfrac{1}{2} + \dfrac{\mu}{\vartheta_{ij} u_{ij}} + \sqrt{\dfrac{1}{4} + \left(\dfrac{\mu}{\vartheta_{ij} u_{ij}}\right)^2} & \text{if } \vartheta_{ij} < 0, \\ \dfrac{1}{2} & \text{if } \vartheta_{ij} = 0, \end{cases}$$

where, for some initial guess y^0 of the dual vector y,

$$\vartheta_{ij} = -y_i^0 + y_j^0 + c_{ij} \, .$$

In PDNET, we set the initial guess

$$y^0 = \frac{\max\{|\, c_{ij} \,| : (i, j) \in \mathcal{E}\}}{\max\{|\, b_i \,| : i \in \mathcal{V}\}} b$$

and parameter

$$\mu = 0.2 \max\{|\, \vartheta_{ij} u_{ij} \,| : (i, j) \in \mathcal{E}\}$$

The primal-dual parameter has an initial value $\mu_0 = \beta_1 \mu$, where in PDNET $\beta_1 = 0.1$. Subsequently, for iterations $k \geq 1$, μ_k is computed as in (7).

The stepsize parameters ϱ_p and ϱ_d are both set to 0.995 throughout the iterations, slightly more conservative than as suggested by [9].

4. Fortran subroutines

The current implementation PDNET consists of a collection of core subroutines with additional subsidiary modules written in Fortran [7]. The software distribution associated with this paper provides fully functional utilities by including Fortran reference implementations for the main program, providing default parameter settings and supplying additional routines for data input and output functions. In this section, we describe the usage of the PDNET framework with comments on user extensions. Specific instructions for building and installing PDNET from the source code are provided with the software distribution. Table 1 lists all modules provided in the PDNET software distribution.

Table 1 – Source code files.

File Name	Description
`dblas1.f`	Fortran Level 1 BLAS reference implementation
`dnsubs.f`	Fortran implementation of Dinic's Algorithm
`fdriver.f90`	Fortran main program
`pdnet_default.f90`	Fortran function setting parameter defaults
`pdnet_maxflow.f90`	Fortran subroutine for invoking maximum flow computation
`pdnet_feasible.f90`	Fortran subroutine for network feasibility checking
`pdnet.f90`	PDNET core subroutines
`pdnet_solreport.f90`	PDNET report generator
`pdnet_read.f90`	Fortran functions for reading network data

We adopted several design and programming style guidelines in implementing the PDNET routines in Fortran. We required that all arrays and variables be passed as subroutine arguments, avoiding the use of `COMMON` blocks. The resulting code is expected to compile and run without modification under most software development and computing environments.

4.1 PDNET core subroutines

The PDNET core subroutines are invoked via a single interface provided by subroutine `pdnet()`. Following the specifications listed in Table 2, the calling program must provide data via the input arguments and allocate the internal and output arrays appropriately as described in Subsection 4.3 in file `fdriver.f90`.

We provide reference implementations of the PDNET main program which also serve as guides for developing custom applications invoking the PDNET core subroutines. In Subsection 4.2, we discuss in detail the input and output routines used in the reference implementations. Subsection 4.4 discusses the setting of parameters in PDNET. In addition, the core subroutines call an external function for maximum flow computation, which is provided in a subsidiary module, with its interface discussed in Subsection 4.5.

4.2 Data input

Programs invoking the PDNET subroutines must supply an instance for the minimum-cost flow problem. As presented in Table 3, the data structure includes the description of the underlying graph, node netflow values, arc costs, capacities and lower bounds. All node and arc attributes are integer valued.

Table 2 – Arguments for `pdnet()`.

Variable	Size	Type	Status	description
b	nn	integer	input	netflow values for each node
c	na	integer	input	cost for each arc
dinfo	34	double precision	input	parameters and statistics
endn	na	integer	input	end node for each arc
info	23	integer	input	parameters and statistics
l	na	integer	input	lower bound of flow for each arc
na	-	integer	input	number of arcs
nn	-	integer	input	number of nodes
optflo	na	integer	output	optimal flow for each arc
strn	na	integer	input	start node for each arc
u	na	integer	input	upper bound of flow for each arc

Table 3 – Network data structure.

Variable	Dimension	Type	description
na	-	integer	number of nodes
nn	-	integer	number of arcs
strn	na	integer	start node for each arc
endn	na	integer	end node for each arc
b	nn	integer	flow values for each node
c	na	integer	cost for each arc
l	na	integer	lower bound of flow for each arc
u	na	integer	upper bound of flow for each arc

The reference implementations of the main program PDNET reads network data from an input file by invoking functions available in the Fortran module `pdnet_read.f90`. These functions build PDNET input data structures from data files in the DIMACS format [2] with instances of minimum cost flows problems. As illustrated in the PDNET module `fdriver.f90`, we provide specialized interfaces used in Fortran programs using dynamic memory allocation.

4.3 Memory allocation

In PDNET, total memory utilization is carefully managed by allocating each individual array to a temporary vector passed as argument to internal PDNET functions. Furthermore, input and output arrays, presented in Table 4, are passed as arguments to subroutine `pdnet()` and must be allocated by the calling procedure.

Table 4 – External arrays.

variable	dimension	type	description
dinfo	34	double precision	parameters and statistics
info	23	integer	parameters and statistics
optflo	na	integer	optimal flow for each arc

4.4 Parameter setting

PDNET execution is controlled by a variety of parameters, selecting features of the underlying algorithm and the interface with the calling program. A subset of these parameters is exposed to the calling procedure via a PDNET core subroutine. The remaining parameters are set at compile time with default values assigned in inside module `pdnet_default.f90`.

The run time parameters listed in Tables 5 and 6 are set with Fortran function `pdnet_setintparm()`. The integer parameters are assigned to components of vector `info` and double precision parameters are assigned to vector `dinfo`.

4.5 Maximum flow computation

PDNET includes a maximum flow computation module called `pdnet_maxflow`, featuring the implementation of Goldfarb and Grigoriadis [6] of Dinic's algorithm, that is used to check the maximum flow stopping criterion. Furthermore, a modification of this module, called `pdnet_feasible`, is called by the module `pdnet_checkfeas()` after reading the network file to compute a maximum flow on the network, therefore checking for infeasibility.

4.6 Running PDNET

Module `fdriver` inputs the network structure and passes control to module `pdnet()`. This subroutine starts by reading the control parameters from the vector `info`, with `pdnet_getinfo()`. Correctness of the input is checked with `pdnet_check()`, and data structure created with `pdnet_datastruct()`. Internal parameters for methods are set with `pdnet_default()`, and additional structures are created with `pdnet_buildstruct()`. Subroutines `pdnet_transform()` and `pdnet_perturb()` are called in order to shift the lower bounds to zero and to transform the data into double precision, respectively. Subroutines `pdnet_probdata()` and `pdnet_checkfeas()` check the problem characteristics and inquire if the network has enough capacity to transport the proposed amount of commodity. The primal-dual main loop is then started and the right-hand-side of the search direction system is computed by `pdnet_comprhs()`. The maximum spanning tree is computed by `pdnet_heap()`, and its optimality is tested by `pdnet_optcheck()`. Under certain conditions ($\mu < 1$), the maxflow stopping criterion is invoked by subroutine `pdnet_checkmaxflow()`. If required, preconditioner switch takes place, followed by a call to `pdnet_precconjgrd()` to solve the Newton direction linear system using the chosen preconditioner. A summary of the iteration is printed by `pdnet_printout()`. Primal and dual updates are made by `pdnet_updatesol()` and stopping criteria check takes place, before returning to the start of the iteration loop.

We now present a usage example. Let us consider the problem of finding the minimum cost flow on the network represented in Figure 2. In this figure, the integers close to the nodes represent the node's produced (positive value) or consumed (negative value) net flows, and the integers close to the arcs stand for their unit cost. Furthermore the capacity of each arc is bounded between 0 and 10. In Figure 3, we show the DIMACS format representation of this problem, stored in file `test.min`. Furthermore, Figures 4 and 5 show the beginning and the end of the printout given by PDNET, respectively.

Table 5 – List of runtime integer parameters assigned to vector `info`.

component	variable	description
1	`bound`	used for building maximum spanning tree
2	`iprcnd`	preconditioner used
3	`iout`	output file unit (6: standard output)
4	`maxit`	maximum number of primal-dual iterations
5	`mxcgit`	maximum number of PCG iterations
6	`mxfv`	vertices on optimal flow network
7	`mxfe`	edges on optimal flow network
8	`nbound`	used for computing additional data structures
9	`nbuk`	number of buckets
10	`pduit`	number of primal-dual iterations performed
11	`root`	sets beginning of structure
12	`opt`	flags optimal flow
13	`pcgit`	number of PCG iterations performed
14	`pf`	iteration level
15	`ierror`	flags errors in input
16	`cgstop`	flags how PCG stopped
17	`mfstat`	maxflow status
18	`ioflag`	controls the output level of the summary
19	`prttyp`	printout level
20	`optgap`	duality gap optimality indicator flag
21	`optmf`	maximum flow optimality indicator flag
22	`optst`	spanning tree optimality indicator flag
23	`mr`	estimate of maximum number of iterations

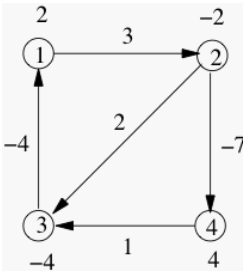

Figure 2 – Example of a minimum cost flow network.

Table 6 – List of runtime double precision parameters assigned to vector `dinfo`.

component	variable	description
1	`dobj`	dual objective function value
2	`dof`	dual objective function value on tree
3	`factor`	factor for Newton step
4	`fcttol`	factorization zero tolerance
5	`gap`	value of the gap
6	`grho0`	factor for `rho0` update
7	`gtolcg`	factor for PCG tolerance
8	`gtolmf`	update factor for maxflow tolerances
9	`mfdopt`	maxflow value
10	`miu`	value of interior-point μ parameter
11	`mntlcg`	lower bound for PCG tolerance
12	`mrho0`	lower bound for `rho0`
13	`oldtol`	value of residual in previous PCG iteration
14	`pcgres`	value of residual in current PCG iteration
15	`pobj`	objective function value
16	`pof`	primal objective function value on tree
17	`rho0`	parameter from IQRD preconditioner
18	`s1fctr`	factor for `miu`
19	`stptol`	tolerance for dual slacks
20	`stpval`	value of largest slack on dual face
21	`tol1`	lower bound of tolerance for maxflow
22	`tol2`	upper bound of tolerance for maxflow
23	`tolcg`	tolerance for PCG stopping criterion
24	`tolcg0`	guess for tolerance for PCG stopping criterion
25	`tolslk`	zero for dual slacks
26	`huge`	largest real number
27	`tolpcg`	initial value for `tolcg`
28	`oldcos`	value of cosine in previous PCG iteration
29	`pcgcos`	value of cosine in current PCG iteration
30	`zero`	value for zero
31	`tolsw1`	maximum value for `sw1`
32	`tolsw2`	maximum value for `sw2`
33	`sw1`	iter limit on PCG for diagonal to spanning tree switch
34	`sw2`	iter limit on PCG for spanning tree to IQRD switch

```
p min 4 5
n 1  2
n 2 -2
n 3 -4
n 4  4
a 1 2 0 10  3
a 2 4 0 10 -7
a 4 3 0 10  1
a 3 1 0 10 -4
a 2 3 0 10  2
```

Figure 3 – File `test.min` with the DIMACS format representation of the problem of Fig. 2.

```
------------------------------------------------------------------------------
PDNET 1.0      Release date: 2002-02-01
Primal-feasible dual-infeasible interior point network flow method

Problem Size and Memory Requirements:

          Number of nodes:              4  Maximum number of nodes:         10000
           Number of arcs:              5   Maximum number of arcs:        200000
       Size of work array:            222        Maximum work array:       4000000

------------------------------------------------------------------------------
Runtime Parameters

       CG Preconditioner:              3             Output level:              1
      Maximum iterations:           1000    Maximum CG iterations:           1000
   Duality Gap Optimality:              1     Max. Flow Optimality:              1
    Span. Tree Optimality:              1

------------------------------------------------------------------------------
Compile-time Parameters

       Output file unit:              6
      Spanning tree root:             1  Number of sort buckets:              5
  Factor for Newton step:0.90000000E+00  Number of sort buckets:0.70064923E-44
------------------------------------------------------------------------------
 CONV>      itr:                1        mu: 0.17152671E+01

 COST>     pobj: -.35209131E+02     dobj: -.69456608E+02  pd-gap: 0.42881677E+02

   CG>  precnd: SPAN              cgstop:  COS              pcgitr:              3
   CG>  pcgtol: 0.16061519E+01    pcgres: 0.47184479E-15
   CG>   cgtol: 0.10000000E-02    pcgcos: 0.22204460E-15

 TREE> pb-feas: TRUE              pb-opt: FALSE
 TREE>     pobj: -.32000000E+02      dobj: -.37683242E+02

MFLOW>  mfstat: INACTIVE
------------------------------------------------------------------------------
 CONV>      itr:                2        mu: 0.57549545E+00

 COST>     pobj: -.25487418E+02     dobj: -.45470504E+02  pd-gap: 0.14387386E+02

   CG>  precnd: SPAN              cgstop:  COS              pcgitr:              2
   CG>  pcgtol: 0.34821594E+00    pcgres: 0.51611112E-01
   CG>   cgtol: 0.10000000E-02    pcgcos: 0.54634098E-04

 TREE> pb-feas: TRUE              pb-opt: FALSE
 TREE>     pobj: -.32000000E+02      dobj: -.35834791E+02

MFLOW>     mxfv:                6     mxfe:                7
MFLOW> pdlotol: 0.95000000E-02  pdhitol: 0.10526316E+03    dobj: -.70000000E+02
MFLOW>  mfstat: SUBOPTIMAL
```

Figure 4 – Startup and first two iterations of a typical PDNET session.

```
------------------------------------------------------------------------------
 CONV>      itr:                8       miu: 0.54012752E-03

 COST>     pobj: -.32000034E+02     dobj: -.32011037E+02  pd-gap: 0.13503188E-01

   CG>  precnd: SPAN              cgstop: COS              pcgitr:              1
   CG>  pcgtol: 0.16712951E-03    pcgres: 0.62266830E-04
   CG>   cgtol: 0.97500000E+00    pcgcos: 0.10817880E-09

 TREE> pb-feas: TRUE              pb-opt: TRUE
 TREE>      pof: -.32000000E+02       dof: -.32003820E+02

MFLOW>     mxfv:                6     mxfe:                7
MFLOW> pdlotol: 0.69833730E-02  pdhitol: 0.14319728E+03    dobj: -.32000000E+02
MFLOW>  mfstat: OPTIMAL

------------------------------------------------------------------------------
PDNET 1.0:  Primal-feasible, dual-infeasible interior point network flow method
OPTIMAL primal flow

                from       to       arc     lower     upper       arc
      arc       node     node      cost     bound     bound      flow

        1          1        2         3         0        10         8
        2          2        4        -7         0        10         6
        3          4        3         1         0        10        10
        4          3        1        -4         0        10         6
```

Figure 5 – Last iteration and results printout in typical PDNET session.

5. Computational results

A preliminary version of PDNET was tested extensively with results reported in [12]. In this section, we report on a limited experiment with the version of PDNET in the software distribution.

The experiments were done on a PC with an Intel Pentium IV processor running at 2 Ghz and 2 Gb of main memory. The operating system is Ubuntu Linux 7.10 (kernel 2.6.22). The code was compiled on the Intel Fortran compiler version 10.0 using the flags `-O3`. CPU times in seconds were computed by calling the Fortran 90 function `cpu_time()`. The test problems are instances of the classes `mesh`, `grid` and `netgen_lo` of minimum-cost network flow problems, taken from the First DIMACS Implementation Challenge [3]. The specifications of the `mesh` instances generated are presented in Table 7. The specifications used in the GRIDGEN generator to build the grid problems are displayed in Table 8. Finally,

the instances of the test set `netgen_lo` were generated according to the guidelines stated in the computational study of Resende and Veiga [15], using the NETGEN generator following the specifications presented in Table 9. All these generators can be downloaded from the FTP site `dimacs.rutgers.edu`.

Table 7 – Specifications for instances in class `mesh`.

Problem	n	m	maxcap	maxcost	seed	x	y	xdeg	ydeg
1	256	4096	16384	4096	270001	36	7	9	5
2	2048	32768	16384	4096	270001	146	14	9	5
3	8192	131072	16384	4096	270001	409	20	9	5

Table 8 – Specifications for instances in class grid.

							arc costs		arc caps	
Problem	nodes	grid size	sources	sinks	average degree	supply	min	max	min	max
1	101	10x10	50	50	16	100000	0	2000	0	2000
2	901	30x30	450	450	16	100000	0	2000	0	2000
3	2501	50x50	1250	1250	16	100000	0	2000	0	2000

Instances of increasing dimension were considered. The three `netgen_lo` instances presented were generated considering $x = 9$, $x = 13$ and $x = 15$ respectively. In Table 10, the number of iterations (IT) and the CPU time (CPU) of PDNET and CPLEX 10 are compared. The reported results show that PDNET tends to outperform CPLEX in the larger instances of the `mesh` and `netgen_lo` set problems, but fails to do so in `grid` set problems. For testset `netgen_lo` the difference is quite noticeable for the largest instance. We observed that in this problem about 90% of the CPU time and iteration count in CPLEX 10 was spent computing a feasible solution. The results in this experience and those reported in [12] show that this code is quite competitive with CPLEX and other efficient network flow codes for large-scale problems.

6. Concluding remarks

In this paper, we describe a Fortran implementation of PDNET, a primal-infeasible dual-feasible interior point method for solving large--scale linear network flow problems. The subroutines are described, and directions for usage are given. A number of technical features of the implementation, which enable the user to control several aspects of the program execution, are also presented. Some computational experience with a number of test problems from the DIMACS collection is reported. These results illustrate the efficiency of PDNET for the solution of linear network flow problems. Source code for the software is available for download at `http://www.research.att.com/~mgcr/pdnet`.

Table 9 – NETGEN specification file for class `netgen_lo`.

`seed`	Random number seed:	27001
`problem`	Problem number (for output):	1
`nodes`	Number of nodes:	$m = 2^x$
`sources`	Number of sources:	2^{x-2}
`sinks`	Number of sinks:	2^{x-2}
`density`	Number of (requested) arcs:	2^{x+3}
`mincost`	Minimum arc cost:	0
`maxcost`	Maximum arc cost:	4096
`supply`	Total supply:	$2^{2(x-2)}$
`tsources`	Transshipment sources:	0
`tsinks`	Transshipment sinks:	0
`hicost`	Skeleton arcs with max cost:	100%
`capacitated`	Capacitated arcs:	100%
`mincap`	Minimum arc capacity:	1
`maxcap`	Maximum arc capacity:	16

Table 10 – Computational experience with instances of the sets `mesh`, `grid` and `netgen_lo` of test problems.

				CPLEX 10		PDNET	
Set	**Problem**	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	**IT**	**CPU**	**IT**	**CPU**
`mesh`	1	256	4096	8383	0.04	40	0.08
	2	2048	32768	108831	2.28	59	1.31
	3	8192	131072	545515	51.06	76	10.20
`grid`	1	101	1616	282	0.01	27	0.02
	2	901	14416	1460	0.03	35	0.36
	3	2501	40016	4604	0.19	44	1.70
`netgen_lo`	1	512	4102	3963	0.03	28	0.08
	2	8192	65709	135028	5.19	46	3.30
	3	32768	262896	800699	112.24	59	27.50

References

(1) Ahuja, N.K.; Magnanti, T.L. & Orlin, J.B. (1993). *Network Flows*. Prentice Hall, Englewood Cliffs, NJ.

(2) DIMACS (1991). The first DIMACS international algorithm implementation challenge: Problem definitions and specifications. World-Wide Web document.

(3) DIMACS (1991). The first DIMACS international algorithm implementation challenge: The benchmark experiments. Technical report, DIMACS, New Brunswick, NJ.

(4) El-Bakry, A.S.; Tapia, R.A. & Zhang, Y. (1994). A study on the use of indicators for identifying zero variables for interior-point methods. *SIAM Review*, **36**, 45-72.

(5) Gay, D.M. (1989). Stopping tests that compute optimal solutions for interior-point linear programming algorithms. Technical report, AT&T Bell Laboratories, Murray Hill, NJ.

(6) Goldfarb, D. & Grigoriadis, M.D. (1988). A computational comparison of the Dinic and network simplex methods for maximum flow. *Annals of Operations Research*, **13**, 83-123.

(7) International Organization for Standardization (1997). Information technology – Programming languages – Fortran – Part 1: Base language. ISO/IEC 1539-1:1997, International Organization for Standardization, Geneva, Switzerland.

(8) Karmarkar, N.K. & Ramakrishnan, K.G. (1991). Computational results of an interior point algorithm for large scale linear programming. *Mathematical Programming*, **52**, 555-586.

(9) McShane, K.A. & Monma, C.L. & Shanno, D.F. (1989). An implementation of a primal-dual interior point method for linear programming. *ORSA Journal on Computing*, **1**, 70-83.

(10) Mehrotra, S. & Ye, Y. (1993). Finding an interior point in the optimal face of linear programs. *Mathematical Programming*, **62**, 497-516.

(11) Portugal, L.; Bastos, F.; Júdice, J.; Paixão, J. & Terlaky, T. (1996). An investigation of interior point algorithms for the linear transportation problem. *SIAM J. Sci. Computing*, **17**, 1202-1223.

(12) Portugal, L.F.; Resende, M.G.C.; Veiga, G. & Júdice, J.J. (2000). A truncated primal-infeasible dual-feasible network interior point method. *Networks*, **35**, 91-108.

(13) Prim, R.C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, **36**, 1389-1401.

(14) Resende, M.G.C. & Veiga, G. (1993). Computing the projection in an interior point algorithm: An experimental comparison. *Investigación Operativa*, **3**, 81-92.

(15) Resende, M.G.C. & Veiga, G. (1993). An efficient implementation of a network interior point method. **In**: *Network Flows and Matching: First DIMACS Implementation Challenge* [edited by David S. Johnson and Catherine C. McGeoch], volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 299-348. American Mathematical Society.

(16) Resende, M.G.C. & Veiga, G. (1993). An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks. *SIAM Journal on Optimization*, **3**, 516-537.

(17) Ye, Y. (1992). On the finite convergence of interior-point algorithms for linear programming. *Mathematical Programming*, **57**, 325-335.

(18) Yeh, Quey-Jen (1989). A reduced dual affine scaling algorithm for solving assignment and transportation problems. PhD thesis, Columbia University, New York, NY.