A TRUNCATED PRIMAL-INFEASIBLE DUAL-FEASIBLE NETWORK INTERIOR POINT METHOD

L.F. PORTUGAL*, M.G.C. RESENDE†, G. VEIGA‡, AND J.J. JÚDICE§

Abstract. In this paper, we introduce the truncated primal-infeasible dual-feasible interior point algorithm for linear programming and describe an implementation of this algorithm for solving the minimum cost network flow problem. In each iteration, the linear system that determines the search direction is computed inexactly, and the norm of the resulting residual vector is used in the stopping criteria of the iterative solver employed for the solution of the system. In the implementation, a preconditioned conjugate gradient method is used as the iterative solver. The details of the implementation are described and the code, PDNET, is tested on a large set of standard minimum cost network flow test problems. Computational results indicate that the implementation is competitive with state-of-the-art network flow codes.

Key words. Interior point method, linear programming, network flows, primal-infeasible dual-feasible, truncated Newton method, conjugate gradient, maximum flow, experimental testing of algorithms.

1. Introduction. The minimum cost network flow problem is one of the most studied problems in optimization, with a wide range of real-world applications [1]. In the past five decades, several computationally efficient algorithms for this problem have been developed. These include the network simplex method [8, 24, 29, 21], the out-of-kilter method [15], the relaxation method [4], and the scaling push-relabel method [19]. Furthermore, work in mathematical programming, stemming from the paper by Karmarkar [27], has generated new algorithms for linear programming that have been specialized to solve network flow problems. In this paper, we describe an efficient implementation of a new interior point network flow method, the truncated primal-infeasible dual-feasible algorithm, and show that this implementation is competitive with previous network flow codes.

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 the following linear program:

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

subject to:

$$\sum_{(i,k)\in\mathcal{E}} x_{ik} - \sum_{(k,j)\in\mathcal{E}} x_{kj} = b_j, \quad j \in \mathcal{V} \tag{1.1}$$

$$l_{ij} \le x_{ij} \le u_{ij}, \quad (i,j) \in \mathcal{E}. \tag{1.2}$$

In this formulation, x_{ij} denotes the flow on edge (i, j) and c_{ij} is the cost of transporting one unit of flow on edge (i, j). For each vertex $j \in \mathcal{V}$, b_j denotes the flow produced or consumed at vertex j. If $b_j > 0$, vertex j is a source. If $b_j < 0$, vertex j is a sink.

* Luis Portugal died on July 21, 1998. At the time of his death, he was affiliated with Departamento de Ciências da Terra, Universidade de Coimbra, 3000 Coimbra, Portugal.

†AT&T Labs Research, Florham Park, NJ 07932 USA

‡AT&T Labs, Lincroft, NJ 07738 USA

§Departamento de Matemática, Universidade de Coimbra, 3000 Coimbra, Portugal

Otherwise ($b_j = 0$), vertex j is a transshipment vertex. For each edge $(i,j) \in \mathcal{E}$, l_{ij} and u_{ij} denote the lower and upper bounds on flow on edge (i,j), respectively. Most often, the problem data are assumed to be integer, and many network flow codes adopt this assumption. However, there can exist applications where the data are real numbers, and algorithms should ideally handle problems with real data.

Constraints of type (1.1) are referred to as the flow conservation equations, while constraints of type (1.2) are called the flow capacity constraints. In matrix notation, the above network flow problem can be formulated as a primal linear program of the special form

$$\min \{c^\top x \mid Ax = b;\ x + s = u;\ x, s \geq 0\}, \tag{1.3}$$

where 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. The dual of (1.3) can be written as:

$$\max \{b^\top y - u^\top w \mid A^\top y - w + z = c;\ z, w \geq 0\}, \tag{1.4}$$

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

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

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

When it is further required that the flow x_{ij} be integer, (1.2) is replaced with

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

Since the vertex-edge incidence matrix A is totally unimodular, when the data is integer all vertex solutions of the linear program are integer. In certain types of network flow problems, such as the assignment problem, one may be only interested in solutions having integer flows, since fractional flows do not have a logical interpretation. An algorithm that finds a vertex solution, such as the simplex method, will necessarily produce an integer optimal flow if the input data is integer. Interior point methods do not necessarily produce integer solutions. However, interior point network flow methods use termination procedures that produce vertex solutions.

In the remainder of this paper 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 can transform the original problem into an equivalent one with $l_{ij} = 0$ for all $(i,j) \in \mathcal{E}$. The case where $c = 0$ is a simple feasibility problem, and can be handled by solving a maximum flow problem [1].

In the last two decades, many approaches have been developed for solving large-scale network flow problems. A history of computational approaches up to 1977 is summarized in [6]. In addition, several computational studies [17, 21, 29, 44] established the fact that specialized network simplex algorithms were orders of magnitude

faster than the best linear programming codes of that time. A collection of FORTRAN codes of efficient algorithms of that period can be found in [55]. Another important class of network optimization algorithms and codes are the relaxation methods described in [4]. More recent computational research is reported in [23] and [19].

In 1984, N. Karmarkar [27] introduced a new polynomial time algorithm for solving linear programming problems. This algorithm and many of its variants, known as interior point methods, have been used to efficiently solve network flow problems. These interior point methods share the same basic computational structure. They begin from an initial interior solution, and at each iteration compute a direction on which some potential function improves. The computation of this direction (the m-vector δ) involves solving a system of linear equations of the type

$$A\Theta A^\top \delta = \beta,$$

where A is the $m \times n$ constraint matrix of the linear program, Θ is an $n \times n$ diagonal scaling matrix, and β is an m-vector.

Early attempts to apply interior point methods to network flow problems can be found in Aronson et al. [3], Karmarkar and Ramakrishnan [28], Rajan [48], Armacost and Mehrotra [2], Yeh [61], Resende and Veiga [51], Resende and Veiga [53], and Kaliski and Ye [26].

In 1991, the first year-long DIMACS algorithm implementation challenge [23] focused on network flow and matching algorithms. Three entries in the network flow portion of the challenge involved interior point approaches. The paper by Joshi, Goldstein, and Vaidya [25] described the implementation and testing of a primal path-following interior point algorithm [59] with a preconditioned conjugate gradient that uses the spanning tree preconditioner, first implemented in [53]. Ramakrishnan, Karmarkar, and Kamath [49], extended the code in [28] to handle assignment problems, by adding modules for initial solution and optimal solution identification. Resende and Veiga [52], described DLNET, an implementation of a dual affine scaling algorithm using diagonal and spanning tree preconditioned conjugate gradient algorithms. All network flow codes were tested on a set of benchmark problems collected during the DIMACS challenge, establishing interior point methods as viable techniques for large-scale linear network flow optimization.

A new indicator to identify the optimal face of network linear programs was introduced in [50] and incorporated in DLNET. Using the new indicator, DLNET was able to find complementary primal-dual integer optimal solutions for all of the DIMACS test instances. The DLNET version used in [52] found solutions that were primal-optimal and not dual-optimal in 4% of the instances. Using a new preconditioner (incomplete QR decomposition), Portugal et al. [45] implemented preconditioned conjugate gradient based dual affine scaling, primal-dual, and predictor corrector interior point algorithms for solving linear transportation problems. On this class of problems, the new preconditioner was shown to outperform both the diagonal and spanning tree preconditioners. A survey of interior point network flow methods is given in [54].

Though the truncated dual affine scaling algorithm is known to perform well in practice on minimum cost network flow problems, there are reasons why one would want to implement a truncated primal-dual interior point method. First, primal-dual methods have been considered by many as the most efficient and robust of all interior point methods for linear programming [33]. We would like to verify if this also holds true for network flow problems. Second, to this date, no efficient implementation of

a truncated primal-dual method has been described [34]. This paper describes such an implementation. Finally, though the dual affine scaling algorithm converges to a strictly complementary solution, there is no theoretical guarantee that its primal iterates converge to a point in the center of the optimal face (the dual iterates do, but the primal iterates can only be guaranteed to converge to a point in the relative interior of the optimal face [11, 43, 58]). Most primal-dual algorithms enjoy this guarantee [22]. This property has been used to develop the robust stopping criteria for the network interior point method described in this paper.

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 $A_{.i}$ and a submatrix of A formed by columns with indices in set S by A_S. Let $x \in \Re^n$. 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 $\|\cdot\|$.

The paper is organized as follows: In Section 2 we present 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. The code, called PDNET, is compared with other efficient implementations of network flow algorithms, namely DLNET [52], CPLEX NETOPT [7], and CS [19], and results are reported in Section 4. Concluding remarks are made in Section 5.

2. Truncated primal-infeasible dual-feasible interior point algorithm. Portugal et al. [45] described the first implementation of a primal-dual interior point network algorithm. The algorithm they implemented, the feasible primal-dual (FPD) algorithm, stemmed from the work of Monteiro and Adler [42], Kojima, Mizuno and Yoshise [31], Megiddo [36], and Tanabe [56]. As its name indicates, the FPD algorithm operates simultaneously on the primal (1.3) and dual problems (1.4). The FPD algorithm requires feasible interior primal and dual starting solutions and iterates in the interior of the feasible region of the primal-dual pair of problems. Let

$$\begin{aligned}\mathcal{Q}_+ &= \{(x,y,s,w,z) \in \Re^{n+m+n+n+n} \ : \ x>0, s>0, w>0, z>0\},\\ \mathcal{S}_+ &= \{(x,y,s,w,z) \in \mathcal{Q}_+ \ : \ Ax=b, \ x+s=u, \ A^\top y - w + z = c\}.\end{aligned}$$

The FPD algorithm begins with a point $(x^0, y^0, s^0, w^0, z^0) \in \mathcal{S}_+$. At iteration k, the Newton direction $(\Delta x^k, \Delta y^k, \Delta s^k, \Delta w^k, \Delta z^k)$ associated with the system of equations that defines the central path [36, 13]

$$\begin{aligned} Ax &= b,\\ x+s &= u,\\ A^\top y - w + z &= c,\\ Xz &= \mu e,\\ Sw &= \mu e,\\ x,s,w,z &\geq 0, \end{aligned} \tag{2.1}$$

for $\mu > 0$, is computed as the solution of the linear system of equations

$$\begin{aligned} A\Delta x^k &= 0,\\ \Delta x^k + \Delta s^k &= 0,\\ A^\top \Delta y^k - \Delta w^k + \Delta z^k &= 0,\\ Z^k \Delta x^k + X^k \Delta z^k &= \mu_k e - X^k Z^k e,\\ W^k \Delta s^k + S^k \Delta w^k &= \mu_k e - W^k S^k e. \end{aligned} \tag{2.2}$$

A new iterate

$$(x^{k+1}, y^{k+1}, s^{k+1}, w^{k+1}, z^{k+1})$$

is computed by moving from $(x^k, y^k, s^k, w^k, z^k)$ in the direction

$$(\Delta x^k, \Delta y^k, \Delta s^k, \Delta w^k, \Delta z^k),$$

as follows

$$\begin{aligned} x^{k+1} &= x^k + \alpha_p \Delta x^k, \\ s^{k+1} &= s^k + \alpha_p \Delta s^k, \\ y^{k+1} &= y^k + \alpha_d \Delta y^k, \\ w^{k+1} &= w^k + \alpha_d \Delta w^k, \\ z^{k+1} &= z^k + \alpha_d \Delta z^k, \end{aligned} \tag{2.3}$$

where α_p and α_d are stepsizes in the primal and dual spaces, respectively. It can be easily verified that the sequence $\{(x^k, y^k, s^k, w^k, z^k)\}$ generated by the FPD method belongs to the set $\mathcal{S}_+$, if stepsizes α_p and α_d are appropriately chosen and the direction components Δx^k, Δy^k, Δs^k, Δw^k, and Δz^k, are computed exactly.

In [45], the solution of the linear system (2.2) is obtained in two steps. First, the Δy^k component of the direction is computed as the solution of the system of normal equations

$$A\Theta^k A^\top \Delta y^k = -A\Theta^k(\mu_k (X^k)^{-1} e - \mu_k (S^k)^{-1} e - c + A^\top y^k), \tag{2.4}$$

where

$$\Theta^k = (Z^k (X^k)^{-1} + W^k (S^k)^{-1})^{-1}. \tag{2.5}$$

The remaining components of the direction are then recovered by

$$\begin{aligned} \Delta x^k &= \Theta^k A^\top \Delta y^k + \Theta^k(\mu_k (X^k)^{-1} e - \mu_k (S^k)^{-1} e - c + A^\top y^k), \\ \Delta s^k &= -\Delta x^k, \\ \Delta z^k &= -z^k + \mu_k (X^k)^{-1} e - Z^k (X^k)^{-1} \Delta x^k, \\ \Delta w^k &= -w^k + \mu_k (S^k)^{-1} e - W^k (S^k)^{-1} \Delta s^k. \end{aligned} \tag{2.6}$$

The practical implementation of the FPD algorithm, presented in [45], differs slightly from the theoretical polynomial-time variant. The main differences concern updating the parameter μ_k and selection of the stepsizes α_p and α_d. In the practical implementation,

$$\mu_k = \beta_1 \frac{(x^k)^\top z^k + (w^k)^\top s^k}{2n} \tag{2.7}$$

with $\beta_1 = 0.1$ and

$$\begin{aligned} \alpha_p &= \varrho_p \ \max\{\alpha \ : \ x^k + \alpha \Delta x^k \geq 0, \ s^k + \alpha \Delta s^k \geq 0\}, \\ \alpha_d &= \varrho_d \ \max\{\alpha \ : \ w^k + \alpha \Delta w^k \geq 0, \ z^k + \alpha \Delta z^k \geq 0\}, \end{aligned} \tag{2.8}$$

where, as suggested in McShane, Monma and Shanno [35], $\varrho_p = \varrho_d = 0.9995$.

The experimental work of McShane, Monma and Shanno [35] and Mehrotra [37] has shown that the FPD algorithm performs well on certain types of linear programs. However, since the FPD algorithm requires the iterates to be both primal and dual feasible, the system of normal equations (2.4) demands a solution having a high degree of accuracy, such as the one computed with direct factorization. In fact, when an iterative method was used in [45], the convergence tolerance of the preconditioned conjugate gradient (PCG) algorithm had to be tightened to such a degree that the algorithm was not competitive with the network simplex code NETFLO [29]. In Portugal et al. [45], some ideas were pointed out to overcome this drawback, leading us to the truncated primal-infeasible dual-feasible algorithm, that we introduce next.

In truncated mathematical programming algorithms, the search direction is not computed exactly. These methods have been shown to be computationally attractive in many instances [9, 10]. Suppose that we want to apply an iterative procedure to approximately solve the system of normal equations that arises in interior point methods. It is natural to ask how approximately can the system be solved without losing the global convergence of the interior point algorithm. Since, in the FPD method, both primal and dual feasibility need to be maintained throughout the iterations, truncated methods cannot be applied. In this paper, we propose a method that is primal infeasible and dual feasible, that we call the *truncated primal-infeasible dual-feasible* (TPIDF) algorithm. This algorithm is presented next.

Let

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

The TPIDF algorithm starts with any solution $(x^0, y^0, s^0, w^0, z^0) \in \bar{\mathcal{S}}_+$. At iteration k, parameter μ_k is computed by (2.7) with $0 < \beta_1 < 1$. 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

$$\begin{aligned} A\Delta x^k &= b - Ax^k + r^k, \\ \Delta x^k + \Delta s^k &= 0, \\ A^\top \Delta y^k - \Delta w^k + \Delta z^k &= 0, \\ Z^k \Delta x^k + X^k \Delta z^k &= \mu_k e - X^k Z^k e, \\ W^k \Delta s^k + S^k \Delta w^k &= \mu_k e - W^k S^k e, \end{aligned} \tag{2.9}$$

where r^k is such that

$$\|r^k\| \leq \beta_0 \|Ax^k - b\|, \ 0 \leq \beta_0 < \beta_1. \tag{2.10}$$

Finally, 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 (2.3).

The solution of the linear system (2.9) is again obtained in two steps. First, we compute the Δy^k component of the direction as the *approximate* solution of the system of normal equations

$$\begin{aligned} A\Theta^k A^\top \Delta y^k = -A\Theta^k(\mu_k (X^k)^{-1} e - \mu_k (S^k)^{-1} e - c + A^\top y^k) + \\ (b - Ax^k), \end{aligned} \tag{2.11}$$

where Θ_k is given by (2.5). Then, the remaining components of the direction are recovered by (2.6).

In studying the global convergence for the TPIDF algorithm, we make use of the theory of Kojima, Megiddo and Mizuno [30]. We can carry out the same analysis by computing the *exact* solution of

$$A\Theta^k A^\top \Delta y^k = -A\Theta^k(\mu_k(X^k)^{-1}e - \mu_k(S^k)^{-1}e - c + A^\top y^k) + (b - Ax^k) + r^k, \tag{2.12}$$

for some r^k verifying (2.10). Let $\bar{\beta}$, γ, and γ_p be such that $0 < \bar{\beta} < 1$, $0 < \gamma < 1$, and $\gamma_p > 0$, and suppose that ε_c and ε_p represent tolerances required for the complementarity gap and primal feasibility, respectively. Kojima, Megiddo, and Mizuno [30] prove that if the sequence $\{(x^k, s^k, y^k, w^k, z^k)\}$ generated by the FPD algorithm is restricted to verify

$$(x^k, s^k, y^k, w^k, z^k) \in \mathcal{G}, \tag{2.13}$$

$$(x^{k+1})^\top z^{k+1} + (w^{k+1})^\top s^{k+1} \leq \bar{\beta}((x^k)^\top z^k + (w^k)^\top s^k), \tag{2.14}$$

where

$$\begin{aligned}\mathcal{G} = \{(x,s,y,w,z) \in \bar{\mathcal{S}}_+ \;\; & x_i z_i \geq \gamma(x^\top z + w^\top s)/2n \; (i = 1,2,\ldots,n), \\ & w_i s_i \geq \gamma(x^\top z + w^\top s)/2n \; (i = 1,2,\ldots,n), \\ & x^\top z + w^\top s \geq \gamma_p \|Ax - b\| \;\; \text{or} \;\; \|Ax - b\| \leq \varepsilon_p\},\end{aligned}$$

is a neighborhood of the central path (2.1), there is an iteration k such that $(x^k, s^k, y^k, w^k, z^k)$ is either an approximate solution satisfying

$$(x^k)^\top z^k + (s^k)^\top w^k \leq \varepsilon_c \quad \text{and} \quad \|Ax^k - b\| \leq \varepsilon_p, \tag{2.15}$$

or verifies

$$\|(w^k, z^k)\|_1 > \omega^*, \tag{2.16}$$

for a large w^*.

Since $\|r^k\|$ is bounded from above, the iterates of the TPIDF algorithm can be restricted in the same manner, and this result also holds. Conditions (2.13–2.14) hold in each iteration if α_p and α_d are chosen in the same way as in [30]. Either the TIPFD algorithm finds a solution satisfying (2.15) or termination criterion (2.16) occurs. In the former case, an approximate optimal solution of both the primal and dual linear programs is reached. In the latter case, Kojima, Megiddo and Mizuno [30] show that a minor variation of the FPD algorithm guarantees that the problem is infeasible. To apply this result for the TPIDF algorithm, we impose the additional assumption that the iterates are bounded, precluding condition (2.16). This assumption, commonly used in the analysis of nonlinear programming algorithms, is reasonable when solving feasible linear programs, especially in the case of network flows, where detection of infeasibility is straightforward.

3. Implementation. In this section, we describe PDNET, an implementation of the truncated primal-infeasible dual-feasible algorithm for solving network flow problems. The Newton direction is computed using a preconditioned conjugate gradient method [40]. Deviating from the theoretical algorithm, the starting solution, as well as the iterates are not restricted to verifying (2.13-2.14). The practical implementation of PDNET uses long feasible stepsizes α_p and α_d given in (2.8) rather than the

stepsizes described in Kojima, Megiddo and Mizuno [30]. Furthermore, to stop the preconditioned conjugate gradient algorithm, we use a stopping criterion based on (2.10) for initial iterations of the TPIDF algorithm. To ensure practical performance, on the later iterations of the TPIDF algorithm, we use the so-called *cosine* rule [53] which does not necessarily satisfy (2.10).

PDNET is written almost entirely in Fortran. C code is used only to read input files in the DIMACS network flow format [23] and for preparing the restricted and augmented networks in the maximum flow stopping criterion of Resende and Veiga [52], described in Subsection 3.2. That C code was borrowed from DLNET.

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. Perhaps the most fundamental requirement of an interior point implementation for network flows is an efficient implementation of an iterative method to compute the so-called interior point search direction at each iteration. Direct methods have been shown to perform poorly on even small instances of these problems [51]. In PDNET, the preconditioned conjugate gradient algorithm is used to solve the linear system

$$M^{-1}(A\Theta^k A^\top)\Delta y^k = M^{-1}\bar{b}, \tag{3.1}$$

where M is a positive definite matrix and Θ^k is given by (2.5), and

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

The aim is to make the preconditioned matrix

$$M^{-1}(A\Theta^k A^\top) \tag{3.2}$$

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 satisfiable direction. In the previous section, we set conditions on the error in the computation of the direction . Later, we discuss some implementation details.

Pseudo-code for the preconditioned conjugate gradient algorithm implemented in PDNET is presented in Figure 3.1. With the exception of the starting point, this is the algorithm implemented in [52, 45]. 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. They can be computed with n additions, $2n$ subtractions, and n multiplications. 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. This is done with the expectation that Θ^k and $\bar{b}$ change little between consecutive interior point iterations, and consequently the direction produced in an iteration should be close to the one produced in the previous iteration. The first time `pcg` is called, $\Delta y_0 = (0, \ldots, 0)$.

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

$$Mz_{i+1} = r_{i+1}, \tag{3.3}$$

is solved. A preconditioner must be such that (3.3) is solved efficiently, while at the same time (3.2) is well conditioned, resulting in few conjugate gradient iterations.

PDNET uses primal-dual variants of the diagonal and spanning tree preconditioners described in [53, 52].

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 $q_i := A\Theta^k A^\top p_i$;
8 $\alpha_i := z_i^\top r_i / p_i^\top q_i$;
9 $\Delta y_{i+1} := \Delta y_i + \alpha_i p_i$;
10 $r_{i+1} := r_i - \alpha_i q_i$;
11 $z_{i+1} := M^{-1} r_{i+1}$;
12 $\beta_i := z_{i+1}^\top r_{i+1} / z_i^\top r_i$;
13 $p_{i+1} := z_{i+1} + \beta_i p_i$;
14 $i := i + 1$
15 **od**;
16 $\Delta y := \Delta y_i$
end pcg;

FIG. 3.1. *The preconditioned conjugate gradient algorithm [40]*

The diagonal preconditioner, $M = \mathrm{diag}(A\Theta^k A^\top)$, 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 [51, 53, 61, 52, 45].

The spanning tree preconditioner was introduced in [53] and used in several codes, e.g. [26, 52, 25, 45]. In that preconditioner, 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. Kruskal's algorithm [32], implemented with the data structures in [57] has been applied to compute the maximal spanning tree, using edges ordered approximately with a bucket sort [25, 52], or exactly using a hybrid QuickSort [26]. In PDNET, an exact maximal spanning tree is computed with the Fibonacci heap variant of Prim's algorithm [47], as described in [1]. This is the code used in [45]. 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

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

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

$$M z_{i+1} = r_{i+1}$$

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.

In PDNET, a heuristic similar to the one implemented in DLNET 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 500 conjugate gradient iterations is imposed.

In Section 2, stopping criterion (2.10) for the conjugate gradient method, based on the primal feasibility residual, was described. In PDNET, that rule, along with a stopping criterion implemented in [52] are implemented. The first stopping criterion monitors the residual of the system of normal equations. Let r_i be the residual at the i-th conjugate gradient iteration and x^k be the primal interior point solution at the k-th interior point iteration. If

$$\|r_i\| \leq \beta_0 \|Ax^k - b\|,$$

the first stopping criterion is triggered. In PDNET, $\beta_0 = 0.0999$, since (2.10) must be satisfied and $\beta_1 = 0.1$. This criterion is always used in the first iterations of the interior point algorithm. When $\|Ax^k - b\|$ becomes small, then it is advisable to terminate the conjugate gradient iterations before this stopping rule is satisfied, since it may be to too conservative. In this respect, PDNET also uses the so-called cosine stopping rule described in [28, 52]. The implementation of the cosine rule in PDNET is described next.

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 $|1 - \cos\theta| < \epsilon_{cos}^k$, where ϵ_{cos}^k is the tolerance at interior point iteration k. PDNET initially uses $\epsilon_{cos}^0 = 10^{-3}$ and, as in Joshi, Goldstein, and Vaidya [25] (for a different stopping criterion), tightens the tolerance after each interior point iteration k, as follows:

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

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. One way to proceed is to compute the cosine every l_{cos} conjugate gradient iterations, as was done in [52]. A more efficient procedure follows from the observation, made in [45], that $(A\Theta^k A^\top)\Delta y_i$ is approximately equal

to $\bar{b}-r_i$, where r_i is the estimate of the residual at the i-th conjugate gradient iteration. Using this approximation,

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

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, in PDNET it is computed at each conjugate gradient iteration.

3.2. Stopping criteria for interior point method. In [52], 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 [52] and [39, 60]. 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

$$\min_{y^*\in\Re^m} \{\|y^* - y^k\| \ : \ A^\top_{\mathcal{F}} y^* = c_{\mathcal{F}}\}.$$

Resende and Veiga [52] 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 $\delta_i = c_i - A^\top_{\cdot i} y^*$. Then,

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

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 [52], 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 [16] and El-Bakry, Tapia, and Zhang [12]. 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

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

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 $i \in \{1, 2, \ldots, n\} \setminus \tilde{\mathcal{F}}$,

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

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

$$A_{\tilde{\mathcal{F}}} x_{\tilde{\mathcal{F}}} = \tilde{b} = b - \sum_{i \in \Omega^+} u_i A_i, \tag{3.4}$$

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

Clearly, from the flow balance constraints (3.4), if a feasible flow $x^*_{\tilde{\mathcal{F}}}$ 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 $\tilde{G} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, where

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

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

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

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 $\tilde{x}_{\tilde{\mathcal{F}}}$ is a feasible flow for the restricted network [52]. 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 and Grigoriadis [20] 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

$$x^0 > 0, \ s^0 > 0, \ w^0 > 0, \ z^0 > 0, \tag{3.5}$$

$$x^0 + s^0 = u, \tag{3.6}$$

and

$$A^\top y^0 - w^0 + z^0 = c, \tag{3.7}$$

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 (2.1), i.e.

$$X^0 z^0 = \mu e \quad \text{and} \quad S^0 w^0 = \mu e, \tag{3.8}$$

for $\mu > 0$.

For $(i, j) \in \mathcal{E}$, let

$$\begin{aligned} x^0_{ij} &= \nu_{ij} u_{ij}, \\ s^0_{ij} &= (1 - \nu_{ij}) \, u_{ij}, \\ z^0_{ij} &= \mu / (\nu_{ij} \, u_{ij}), \\ w^0_{ij} &= \mu / ((1 - \nu_{ij}) \, u_{ij}), \end{aligned}$$

be the starting solution, where $0 < \nu_{ij} < 1$ and $\mu > 0$. It is easy to verify that this starting solution satisfies (3.5–3.6) as well as (3.8).

Condition (3.7) is satisfied if, for $(i, j) \in \mathcal{E}$, ν_{ij} is chosen as

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

```
12 250 MHZ IP19 Processors
CPU: MIPS R4400 Processor Chip Revision:  6.0
FPU: MIPS R4010 Floating Point Chip Revision:  0.0
Data cache size:  16 Kbytes
Instruction cache size:  16 Kbytes
Secondary unified instruction/data cache size:  1 Mbyte
Main memory size:  2560 Mbytes, 2-way interleaved
```

FIG. 4.1. *Hardware configuration (partial output of system command* `hinv`*)*

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

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

This choice has worked well in practice.

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 (2.7).

The stepsize parameters ϱ_p and ϱ_d are both set to 0.995 throughout the iterations, slightly more conservative than as suggested by [35].

4. Experimental results. We next show some performance results for PDNET. We use as the set of test problems in our computational experiment, instances from several classes of pure minimum cost network flow problems, taken from the First DIMACS Implementation Challenge [23], included in the computational study of Resende and Veiga [52]. The problem classes considered in this study are Grid-Density-8, Grid-Density-16, Grid-Long, Grid-Wide, Netgen-Hi, Netgen-Lo, and Mesh-1. A description of the characteristics of these problems is given in [52]. The generators can be retrieved from the ftp site `dimacs.rutgers.edu`. From each problem class, instances of increasing dimension are considered.

PDNET is compared with three efficient network flow implementations: the dual affine scaling interior point network flow code DLNET [52], the network simplex code CPLEX NETOPT [7], and CS [19], an implementation of the push-relabel algorithm. In [52], DLNET was compared with the network simplex code NETFLO [29] and RELAXT-3 [5], an implementation of the relaxation algorithm. The performance of PDNET relative to those two codes can be inferred from the comparison with DLNET.

The parameter settings for PDNET have been described in Section 3. As is the case with any interior point code, PDNET is sensitive to parameter settings. For example, the algorithm can be slowed down by decreasing the step size parameters α_p and α_d or by decreasing the initial cosine parameter ϵ_{cos}^0. Though parameter tuning required a considerable amount of experimentation, it is possible that further tuning may produce even better performance. DLNET version 2.2a was used in the experiments.

The stepsize parameter γ^k is set to 0.99 for the first 10 dual affine scaling iterations and at each iteration, the relative dual objective function improvement

$$\mathcal{I}^k = \frac{(b^\top y^k - u^\top z^k) - (b^\top y^{k-1} - u^\top z^{k-1})}{|b^\top y^k - u^\top z^k|}$$

is computed. After iteration 10, the stepsize is determined as follows:

$$\gamma^k = \begin{cases} 0.95, & \text{if } \mathcal{I}^k \geq 0.0001, \\ 0.66, & \text{if } 0.00001 \leq \mathcal{I}^k < 0.0001, \\ 0.50, & \text{if } \mathcal{I}^k < 0.00001. \end{cases}$$

In practice, DLNET never terminated with a value $\mathcal{I}^k \geq 0.00001$. The indicator function Indicator 3 of [50] was used with parameters $\kappa_0 = 0.7$ and $\kappa_1 = 0.9$. DLNET used diagonal and spanning tree preconditioners and the switching heuristic described in [52]. The PB stopping criterion was checked every iteration on which the spanning tree was computed. The MF criterion was not ever checked during the first 5 iterations. Once $\mathcal{I}^k < 10^{-10}$ the MF was checked every 10 iterations, and when $\mathcal{I}^k < 10^{-12}$ it was checked every 5 iterations. The conjugate gradient cosine parameter ϵ_{cos} was set fixed to 10^{-3} throughout all iterations and the exact cosine rule was checked every 5 conjugate gradient iterations.

The versions of CPLEX NETOPT and CS were 4.0.7 and 1.2, respectively. Both codes were run using default parameter settings.

The experiments were performed on an SGI Challenge computer with the configuration summarized in Figure 4.1. Though this hardware is fast for integer arithmetic, it was not designed for floating point computations. We expect both PDNET and DLNET to benefit from systems with improved floating point performance comparatively to integer arithmetic. For example, the next generation processor in the MIPS series, the R10000, performs almost exactly as the R4400 on integer codes. However, floating point performance benchmarks are 3 to 4 times as fast.

For every class of problems we present a figure with CPU time ratios between DLNET and PDNET, CPLEX and PDNET, and CS and PDNET, and three tables. The first two tables provide similar information about PDNET and DLNET: dimensions of problem solved, number of interior point iterations, total CPU time of the interior point algorithm, total number of conjugate gradient iterations, number of times MF stopping criterion was activated, total CPU time spent in MF stopping criterion routines, and stopping criterion verifying optimality. The third table lists statistics for CPLEX (number of iterations and CPU time) and CS (CPU time).

Figure 4.2 and Tables 4.1–4.3 summarize the runs for problem class Grid-Density-8. For those runs, we make the following remarks:

- Nine instances, having up to 65,536 vertices, were solved by all four codes. We also conducted an experiment comparing PDNET and CS on an instance having 131,072 vertices.
- PDNET required more interior point iterations than DLNET except for the instance with 65,536 vertices. However, the total number of conjugate gradient iterations in PDNET was always smaller than in DLNET.
- The running time taken by PDNET grew slower than that taken by DLNET and PDNET was over 4.75 times faster than DLNET on the instance having 65,636 vertices.
- On the instance having 65,636 vertices, the total number of conjugate gradient iterations required by PDNET was about 2.5 times smaller than that required by DLNET.

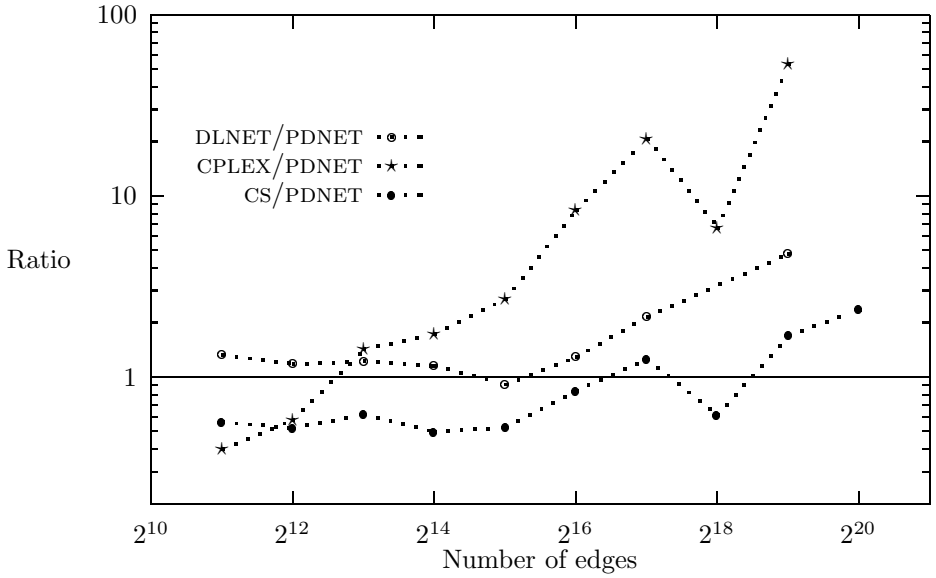

FIG. 4.2. *CPU time ratios for problem class Grid-Density-8*

TABLE 4.1
PDNET *statistics: test problem class Grid-Density-8*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	2048	26	0.50	148	3	0.03	MF
512	4096	33	1.38	165	4	0.09	PB
1024	8192	32	3.10	148	4	0.23	MF
2048	16384	36	9.21	221	4	0.53	MF
4096	32768	43	26.16	278	5	1.43	MF
8192	65536	41	49.39	259	4	3.84	MF
16384	131072	44	114.57	297	5	12.34	MF
32768	262144	80	910.41	2075	19	74.08	MF
65536	524288	51	780.06	456	7	161.37	MF
131072	1048576	56	1969.65	566	6	455.24	MF

- With respect to CPLEX and CS, the CPU time taken by PDNET grew at a slower rate.

Figure 4.3 and Tables 4.4–4.6 summarize the runs for problem class Grid-Density-16. For those runs, we make the following remarks:

- Eight instances, having up to 32,768 vertices, were solved by all four codes. Experiments comparing PDNET with CS were also conducted on instances having 65,536, 131,072 and 263,144 vertices. The density of these networks is twice that of the networks of Grid-Density-8.
- With the exception of three instances, PDNET required less interior point iterations than DLNET. The total number of conjugate gradient iterations was also smaller in PDNET except for one of the instances.
- The running time for PDNET grew slower than for CPLEX and CS, but grew at about the same rate as for DLNET.
- On the instance having 32,768 vertices, the total number of conjugate gradient iterations in PDNET was about 1.5 times smaller than in DLNET.

TABLE 4.2
DLNET *statistics: test problem class Grid-Density-8*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	2048	21	0.66	181	0	0.00	PB
512	4096	26	1.62	192	0	0.00	PB
1024	8192	29	3.78	234	0	0.00	PB
2048	16384	33	10.55	316	0	0.00	PB
4096	32768	32	23.69	376	0	0.00	PB
8192	65536	32	63.82	512	0	0.00	PB
16384	131072	43	245.42	615	0	0.00	PB
65536	524288	65	3708.66	1114	1	149.57	MF

TABLE 4.3
CPLEX *and* CS *statistics: test problem class Grid-Density-8*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
256	2048	4388	0.20	0.28
512	4096	13382	0.80	0.72
1024	8192	34139	4.41	1.92
2048	16384	70121	15.91	4.58
4096	32768	184044	70.93	13.73
8192	65536	391823	411.58	41.32
16384	131072	877322	2366.63	142.80
32768	262144	3627665	6080.27	561.43
65536	524288	4241328	41675.75	1321.80
131072	1048576	DNR	DNR	4616.60

TABLE 4.4
PDNET *statistics: test problem class Grid-Density-16*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	4096	42	1.46	255	4	0.05	MF
512	8192	49	4.18	329	4	0.14	MF
1024	16384	57	11.27	408	5	0.39	MF
2048	32768	64	29.11	508	7	1.41	MF
4096	65536	67	65.26	453	6	2.45	MF
8192	131072	80	166.49	703	8	7.16	MF
16384	262144	97	423.58	890	7	16.05	MF
32768	524288	94	1069.76	858	8	42.68	MF
65536	1048576	113	2610.31	1111	11	255.44	MF
131072	2097152	106	5031.01	1041	9	479.28	MF
262144	4194304	114	16256.70	1735	14	1610.52	MF

TABLE 4.5
DLNET *statistics: test problem class Grid-Density-16*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	4096	32	1.84	251	0	0.00	PB
512	8192	60	7.20	426	1	0.03	MF
1024	16384	65	18.01	515	1	0.09	MF
2048	32768	79	44.74	661	1	0.26	MF
4096	65536	74	98.57	669	1	0.85	MF
8192	131072	99	476.72	1052	1	2.38	MF
16384	262144	94	1034.19	1092	1	10.21	MF
32768	524288	94	2305.82	1223	1	34.28	MF

TABLE 4.6
CPLEX *and* CS *statistics: test problem class Grid-Density-16*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
256	4096	9244	0.43	0.57
512	8192	24725	1.52	1.40
1024	16384	70440	12.16	3.92
2048	32768	148281	37.07	9.85
4096	65536	380078	182.89	31.97
8192	131072	884825	979.60	95.13
16384	262144	2104805	4395.04	281.17
32768	524288	5399631	27203.37	1044.57
65536	1048576	DNR	DNR	2325.98
131072	2097152	DNR	DNR	6177.60
262144	4194304	DNR	DNR	21110.75

TABLE 4.7
PDNET *statistics: test problem class Grid-Long*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
514	1008	23	0.40	155	3	0.05	PB
1026	2000	29	1.12	254	2	0.09	PB
2050	3984	36	4.00	445	3	0.25	MF
4098	7952	45	12.47	731	3	0.54	MF
8194	15888	56	38.60	1075	7	2.63	MF
16386	31760	68	108.15	1577	6	7.55	MF
32770	63504	72	297.51	2376	8	13.73	MF
65538	126992	98	1045.05	3826	18	58.65	MF
131074	253968	102	2424.17	4390	9	77.75	MF

TABLE 4.8
DLNET *statistics: test problem class Grid-Long*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
514	1008	19	0.60	236	0	0.00	PB
1026	2000	29	1.50	402	0	0.00	PB
2050	3984	38	4.73	639	0	0.00	PB
4098	7952	73	17.11	1112	1	0.19	MF
8194	15888	56	42.27	1507	0	0.00	PB
16386	31760	84	166.73	2517	0	0.00	PB
32770	63504	106	470.00	3061	0	0.00	PB
65538	126992	169	2882.79	4228	0	0.00	PB

TABLE 4.9
CPLEX *and* CS *statistics: test problem class Grid-Long*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
514	1008	1152	0.09	0.15
1026	2000	2210	0.21	0.32
2050	3984	4029	0.55	0.62
4098	7952	7775	2.00	1.62
8194	15888	14350	7.69	4.22
16386	31760	29728	28.90	9.02
32770	63504	46894	181.88	24.48
65538	126992	91309	908.00	74.38
131074	253968	175833	4188.19	224.12

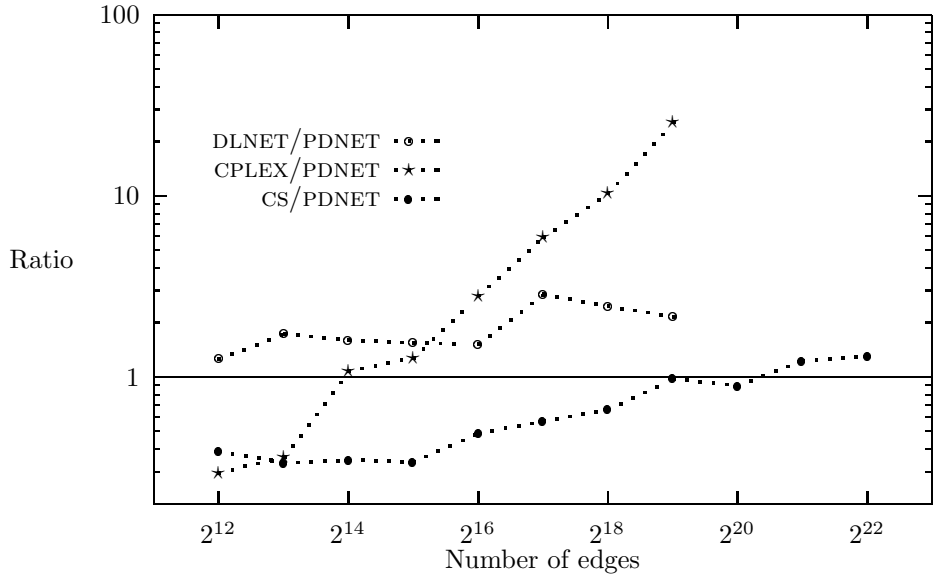

FIG. 4.3. *CPU time ratios for problem class Grid-Density-16*

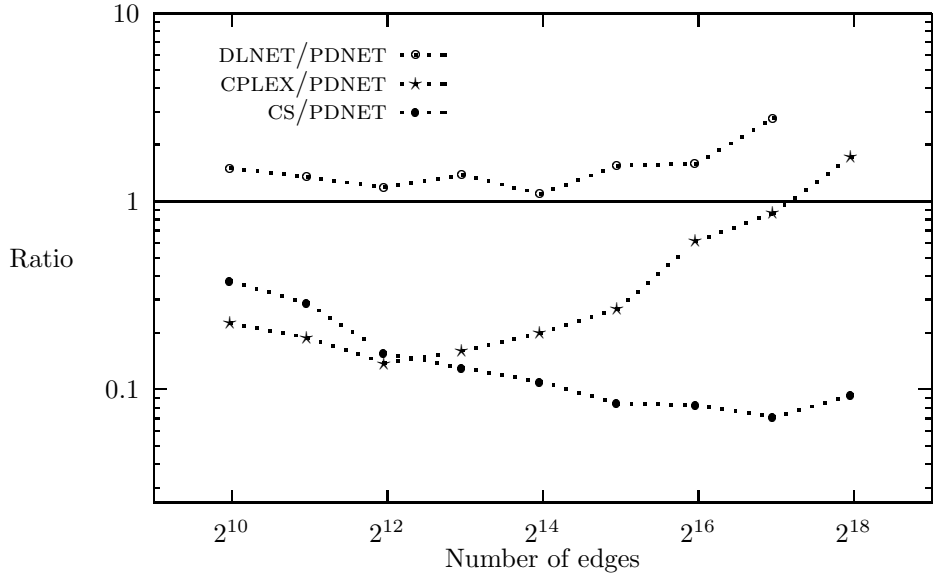

FIG. 4.4. *CPU time ratios for problem class Grid-Long*

Figure 4.4 and Tables 4.7–4.9 summarize the runs for problem class Grid-Long. For those runs, we make the following remarks:

- Eight instances, having up to 65,538 vertices and 126,992 edges, were solved by all four codes. An experiment comparing PDNET with CS and CPLEX NETOPT was conducted on an instance having 131,074 vertices.
- In general, PDNET required less interior point iterations than DLNET. The total number of conjugate gradient iterations in PDNET was smaller by less

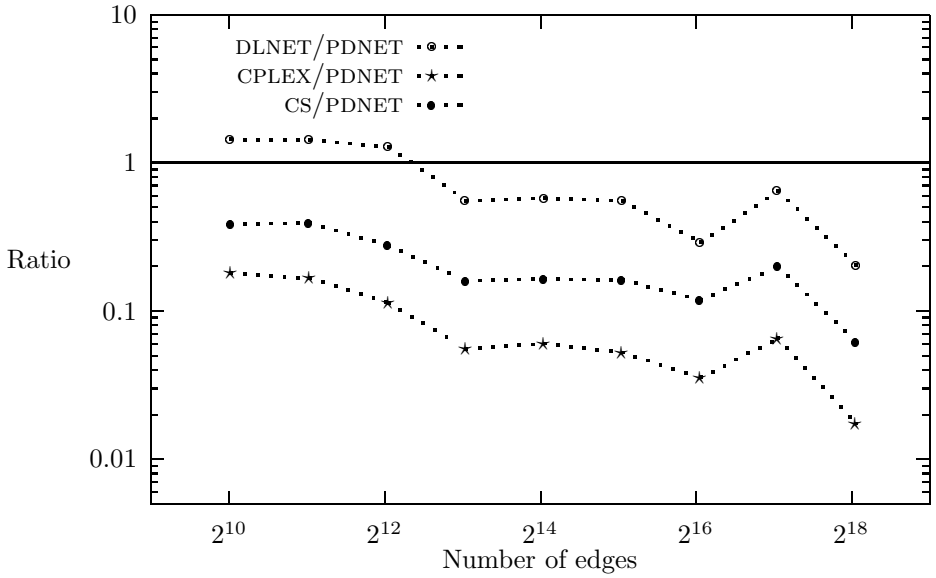

FIG. 4.5. *CPU time ratios for problem class Grid-Wide*

TABLE 4.10
PDNET *statistics: test problem class Grid-Wide*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
514	1040	23	0.39	156	2	0.02	PB
1026	2096	26	1.02	155	2	0.10	MF
2050	4208	41	3.20	238	2	0.17	PB
4098	8432	65	14.76	396	2	0.31	PB
8194	16880	64	34.11	349	3	1.07	PB
16386	33776	86	81.58	407	3	2.18	PB
32770	67568	166	369.88	600	2	3.48	PB
65538	135152	123	629.44	538	6	43.38	MF
131074	270320	450	5261.84	1476	0	0.00	PB

than 60% for all instances.

- The running time taken by PDNET grew slower than those of CPLEX and DLNET, and faster than that of CS.
- On the instance having 65,636 vertices, the total number of conjugate gradient iterations in PDNET was about 10% smaller than in DLNET.

Figure 4.5 and Tables 4.10–4.12 summarize the runs for problem class Grid-Wide. For those runs, we make the following remarks:

- Nine instances, having up to 131,074 vertices and 270,320 edges were solved by all four codes.
- DLNET required slightly more interior point and conjugate gradient iterations than PDNET on the smallest instances. However, on the largest instances, the number of interior point and conjugate gradient iterations was significantly less in DLNET.
- The running time for PDNET grew faster than for the other three codes.
- On the largest instance, the total number of conjugate gradient iterations in PDNET was 3.5 times greater than in DLNET.

TABLE 4.11
DLNET *statistics: test problem class Grid-Wide*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
514	1040	24	0.56	325	0	0.00	PB
1026	2096	30	1.45	391	0	0.00	PB
2050	4208	40	4.10	555	0	0.00	PB
4098	8432	38	8.14	536	0	0.00	PB
8194	16880	43	19.53	581	0	0.00	PB
16386	33776	49	45.10	458	0	0.00	PB
32770	67568	55	107.26	397	0	0.00	PB
65538	135152	71	402.51	388	0	0.00	PB
131074	270320	82	1070.37	411	0	0.00	PB

TABLE 4.12
CPLEX *and* CS *statistics: test problem class Grid-Wide*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
514	1040	1106	0.07	0.15
1026	2096	2448	0.17	0.40
2050	4208	5451	0.36	0.88
4098	8432	10684	0.82	2.35
8194	16880	18867	2.05	5.60
16386	33776	37103	4.28	13.10
32770	67568	75470	13.10	43.87
65538	135152	140813	40.87	124.70
131074	270320	280883	91.41	322.23

TABLE 4.13
PDNET *statistics: test problem class Netgen-Hi*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	2048	31	0.53	150	4	0.02	PB
512	4101	33	1.19	156	3	0.05	MF
1024	8214	41	3.38	179	6	0.19	PB
2048	16414	38	8.22	225	6	0.42	PB
4096	32858	44	24.50	303	10	1.97	PB
8192	65734	52	71.33	462	8	3.47	MF
16384	131409	55	166.50	510	11	13.47	PB
32768	262903	61	470.60	785	9	26.48	MF
65536	525803	68	1557.67	1319	12	81.92	MF

TABLE 4.14
DLNET *statistics: test problem class Netgen-Hi*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	2048	29	0.91	255	0	0.00	PB
512	4101	34	2.03	307	0	0.00	PB
1024	8214	41	6.10	537	0	0.00	PB
2048	16414	49	17.52	596	0	0.00	PB
4096	32858	55	45.51	851	0	0.00	PB
8192	65734	62	106.73	731	0	0.00	PB
16384	131409	74	488.95	2012	0	0.00	PB
32768	262903	92	1746.94	3023	1	18.61	MF
65536	525803	107	8697.92	5153	1	95.01	MF

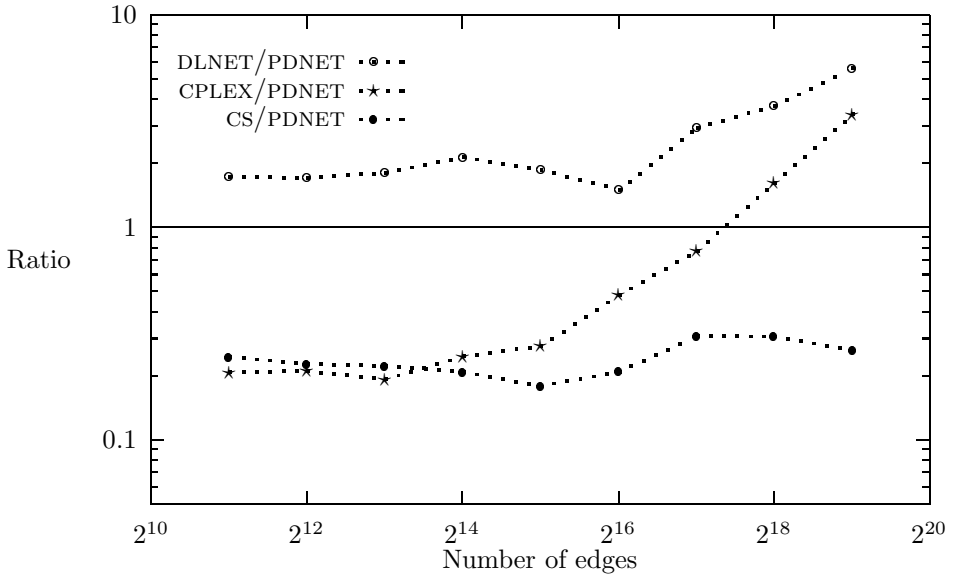

FIG. 4.6. *CPU time ratios for problem class Netgen-Hi*

TABLE 4.15
CPLEX *and* CS *statistics: test problem class Netgen-Hi*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
256	2048	1033	0.11	0.13
512	4101	2855	0.25	0.27
1024	8214	6679	0.65	0.75
2048	16414	11123	2.03	1.70
4096	32858	29138	6.73	4.38
8192	65734	75640	34.13	14.87
16384	131409	137023	129.00	51.32
32768	262903	374410	758.82	144.10
65536	525803	1077281	5264.24	407.68

Figure 4.6 and Tables 4.13–4.15 summarize the runs for problem class Netgen-Hi. For those runs, we make the following remarks:

- Nine instances, having up to 65,538 vertices and 525,803 edges were solved by all four codes.
- With the exception of two small instances, PDNET took fewer iterations than DLNET. DLNET took more conjugate gradient iterations than PDNET in all the instances.
- The growth rate of CPU time for PDNET was smaller than for DLNET and CPLEX and was about the same for CS.
- On the largest instance, the total number of conjugate gradient iterations in PDNET was 4 times smaller than in DLNET.

Figure 4.7 and Tables 4.16–4.18 summarize the runs for problem class Netgen-Lo. For those runs, we make the following remarks:

- Nine instances, having up to 65,538 vertices and 525,803 edges were solved by all four codes.
- PDNET required fewer interior point iterations than DLNET except for two

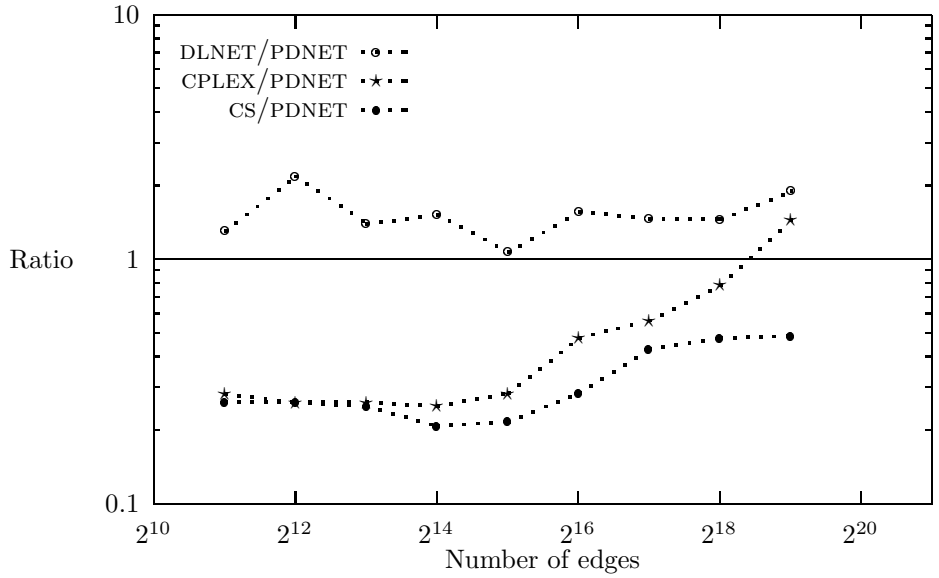

FIG. 4.7. *CPU time ratios for problem class Netgen-Lo*

TABLE 4.16
PDNET *statistics: test problem class Netgen-Lo*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	2048	21	0.50	246	3	0.03	PB
512	4101	26	1.35	302	6	0.10	PB
1024	8214	32	4.08	352	7	0.29	PB
2048	16414	41	13.02	484	10	0.98	PB
4096	32858	44	33.27	607	9	1.96	PB
8192	65734	47	84.72	699	9	4.45	PB
16384	131409	55	234.19	963	10	17.19	MF
32768	262903	60	681.91	1257	12	48.52	MF
65536	525803	60	1766.04	1469	11	147.89	MF

instances. With the exception of the smallest instance, The total number of conjugate gradient iterations was also smaller in PDNET.

- The growth rate of CPU time for PDNET was smaller than for CPLEX and slightly smaller than for CS. PDNET had a growth rate of CPU time similar to DLNET.
- On the largest instance, the total number of conjugate gradient iterations in PDNET was about 10% smaller than in DLNET.

Figure 4.8 and Tables 4.19–4.21 summarize the runs for problem class Mesh-1. For those runs, we make the following remarks:

- Five instances, having up to 65,538 vertices and 131,072 edges were solved by all four codes, with an additional instance having 262,144 vertices solved by PDNET, CPLEX and CS.
- PDNET required fewer interior point iterations than DLNET, except for the smallest instance. The total number of conjugate gradient iterations was always smaller in PDNET.
- The growth rate of CPU time for PDNET was smaller than for CPLEX and

TABLE 4.17
DLNET *statistics: test problem class Netgen-Lo*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	2048	19	0.65	233	0	0.00	PB
512	4101	48	2.94	533	0	0.00	PB
1024	8214	36	5.69	532	0	0.00	PB
2048	16414	52	19.78	803	0	0.00	PB
4096	32858	42	35.56	644	0	0.00	PB
8192	65734	68	132.45	1004	1	1.11	MF
16384	131409	73	342.18	1220	1	6.17	MF
32768	262903	88	993.39	1635	1	20.40	MF
65536	525803	102	3337.98	1685	1	111.35	MF

TABLE 4.18
CPLEX *and* CS *statistics: test problem class Netgen-Lo*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
256	2048	2876	0.14	0.13
512	4101	7171	0.35	0.35
1024	8214	16956	1.06	1.02
2048	16414	29354	3.26	2.70
4096	32858	63856	9.39	7.18
8192	65734	146424	40.25	24.07
16384	131409	259140	131.27	100.05
32768	262903	556433	533.81	322.33
65536	525803	1212399	2549.02	856.62

DLNET. However, the solution time for PDNET grew faster than for CS.

- On the instance with 65,536 vertices, the number of total conjugate gradient iterations in PDNET was 1.5 times smaller than in DLNET.

We conclude this section with the following general remarks:

- Neither PDNET nor DLNET was uniformly better with respect to the number of interior point iterations. The small difference in this number can be attributed to the different optimality indicators used in the MF stopping criterion. This counters the general belief that, for non-truncated methods, the infeasible primal-dual algorithms perform better than the dual affine methods.
- In general, total running times for PDNET were slightly better than those for DLNET, independent of problem size. We observe that the number of conjugate gradient iterations per interior point iteration was smaller for PDNET explaining the improved running times. The difference in the number of inner iterations can be explained by two of the main differences in the implementations:
  - PDNET uses Prim's algorithm to compute an exact maximum weight spanning tree, while DLNET uses Kruskal's method with a bucket sort that computes an approximate maximum weight spanning tree. Therefore, the MST preconditioner is expected to perform better in PDNET than in DLNET.
  - DLNET computes the exact cosine value every other five conjugate gradient iterations while PDNET estimates this value every iteration. Thus, DLNET has the additional cost of computing the exact cosine value and usually performs more conjugate gradient iterations than necessary (four

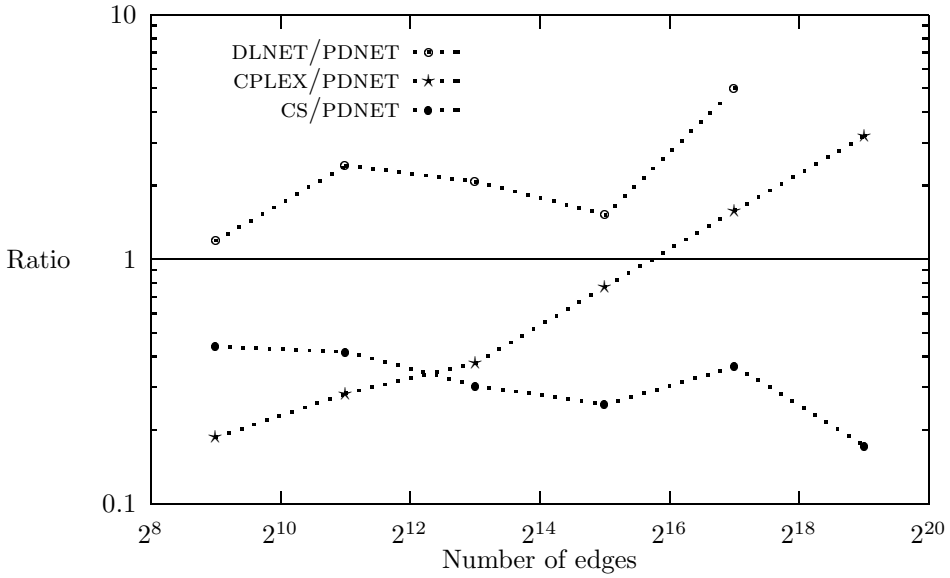

FIG. 4.8. *CPU time ratios for problem class Mesh-1*

TABLE 4.19
PDNET *statistics: test problem class Mesh-1*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	512	17	0.16	109	4	0.03	MF
1024	2048	21	0.96	175	5	0.27	PB
4096	8192	26	7.48	290	7	2.74	PB
16384	32768	30	51.51	477	8	17.00	MF
65536	131072	36	364.51	789	12	178.74	PB
262144	524288	41	4800.78	1396	13	3187.57	MF

in the worst case).

- When compared to CPLEX NETOPT, PDNET was asymptotically faster on all problem classes with the exception of Grid-Long.
- The push-relabel code CS was the overall best performer for the size instances considered in this computational study. However, there is an advantage for PDNET in several classes when we observe the asymptotic growth of CS/PDNET running time ratio. For instance, in problem classes Grid-Density-8 and Grid-Density-16, PDNET displays smaller total running times for the larger instances.

5. Concluding remarks. In this paper, we present the first implementable truncated primal-dual interior point algorithm for linear programming. We also describe a practical primal-infeasible dual-feasible variant of this algorithm PDNET, specialized to minimum cost network flow problems. The code is tested on a standard set of test problems and compared with several efficient network flow codes. No single code was asymptotically the fastest on all classes of problems. PDNET is clearly asymptotically the fastest in two classes of problems (Grid-Density-8 and Grid-Density-16) and is arguably asymptotically the fastest on Netgen-Lo. The other codes tested in our computational study outperformed PDNET for some problem classes. CS was

TABLE 4.20
DLNET *statistics: test problem class Mesh-1*

network size		int point alg		CG	max flow		stopping
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	itrs	calls	time	criterion
256	512	15	0.19	182	0	0.00	PB
1024	2048	36	2.31	454	0	0.00	PB
4096	8192	64	15.47	817	0	0.00	PB
16384	32768	52	78.20	905	0	0.00	PB
65536	131072	83	1820.77	1183	1	105.05	MF

TABLE 4.21
CPLEX *and* CS *statistics: test problem class Mesh-1*

network size		CPLEX NETOPT V4.0.7		CS
$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	itrs	time	time
256	512	787	0.03	0.07
1024	2048	3613	0.27	0.40
4096	8192	16404	2.82	2.27
16384	32768	67299	39.72	13.08
65536	131072	288322	575.66	132.60
262144	524288	1269945	15385.33	825.58

asymptotically the fastest on two problem classes and the network flow code of CPLEX was asymptotically the fastest on one problem class.

The code described in this paper can be further improved. For instance, the technique used for computing the maximum flows in the MF stopping criterion is the implementation of the rather outdated Dinic's algorithm described in [20]. A more modern maximum flow algorithm, such as the push/relabel algorithm of Goldberg and Tarjan [18], will contribute to make PDNET more efficient. The current version of the code does not implement a scheme to fix edges to their upper or lower bounds, as described in [53, 25]. Such a scheme has been shown to improve the efficiency of interior point network flow codes on certain classes of problems. In [53], a parallel implementation of a dual affine scaling network flow algorithm was shown to benefit greatly from the parallel implementation of the conjugate gradient code. We expect PDNET to also benefit from such computer architectures. Finally, since the code involves heavy floating-point computations, it will surely benefit from faster floating-point processors that are becoming available. The newer MIPS R10000 processor executes floating point computations three to four times as fast as the MIPS R4400 used in this study. The interior point codes will benefit from this increase in speedup while most of the other codes, which do little floating-point computations, will not enjoy this speedup.

We further observe that this truncated algorithm can also be used to solve general linear programming problems provided good preconditioner are available. We hope this paper will lead to further research both into the implementation of truncated methods for other special structure mathematical programming problems, as well as theoretical analysis of truncated variants of other interior point techniques.

After the first manuscript of this paper was written in 1994, four related papers have appeared in the literature. In [38], Mehrotra and Wang present a new preconditioner for a preconditioned conjugate gradient based network interior point method. Their preconditioner mimics the diagonal preconditioner used in this paper during the early iterations of the interior point method and gradually changes to mimic the spanning tree preconditioner. Mehrotra and Wang report encouraging computational

results. In [46], Portugal et al. study and compare the preconditioners available in the literature for network interior point methods. Upper bounds for the condition numbers of the preconditioned matrices of the systems of linear equations that define the search directions are derived. The preconditioners are tested using PDNET. A computational comparison of the preconditioners on a set of standard problems is presented. Mizuno and Jarre [41] and Freund et al. [14] describe inexact interior point algorithms for linear programming guaranteeing global convergence by assuming feasibility of the problem. Also a proof of polynomiality is given for the method in [41]. The algorithm in [41] is not a practical algorithm for large sparse problems since it requires a norm to be computed by using an orthogonal decomposition procedure. In Freund et al. [14] it is assumed that a positive lower bound for the smallest singular value of A is available. This bound is used to define the accuracy required in the computation of the search direction.

REFERENCES

[1] N. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] A. Armacost and S. Mehrotra, *Computational comparison of the network simplex method with the affine scaling method*, Opsearch, 28 (1991), pp. 26–43.

[3] J. Aronson, R. Barr, R. Helgason, J. Kennington, A. Loh, and H. Zaki, *The projective transformation algorithm of Karmarkar: A computational experiment with assignment problems*, Tech. Rep. 85-OR-3, Department of Operations Research, Southern Methodist University, Dallas, TX, August 1985.

[4] D. Bertsekas, *Linear network optimization: Algorithms and codes*, MIT Press, Cambridge, MA, 1991.

[5] D. Bertsekas and P. Tseng, *Relaxation methods for minimum cost ordinary and generalized network flow problems*, Operations Research, 36 (1988), pp. 93–114.

[6] G. Bradley, G. Brown, and G. Graves, *Design and implementation of large scale primal transshipment algorithms*, Management Science, 24 (1977), pp. 1–34.

[7] CPLEX Optimization Inc., *Using the CPLEX(TM) callable library and CPLEX(TM) mixed integer library including the CPLEX(TM) linear optimizer and CPLEX(TM) mixed integer optimizer – version 3.0*, 1994.

[8] G. Dantzig, *Application of the simplex method to a transportation problem*, in Activity Analysis of Production and Allocation, T. Koopsmans, ed., John Wiley and Sons, 1951.

[9] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 400–408.

[10] R. S. Dembo and T. Steihaug, *Truncated-Newton algorithms for large scale unconstrained optimization*, Mathematical Programming, 26 (1983), pp. 190–212.

[11] I. Dikin, *Determination of interior point of a system of linear inequalities*, tech. rep., Siberian Energy Institute, Irkutsk, USSR, 1991.

[12] A. El-Bakry, R. Tapia, and Y. Zhang, *A study on the use of indicators for identifying zero variables for interior-point methods*, SIAM Review, 36 (1994), pp. 45–72.

[13] A. Fiacco and G. McCormick, *Nonlinear programming: Sequential unconstrained minimization technique*, Wiley, New York, 1968.

[14] R. W. Freund, F. Jarre, and S. Mizuno, *Convergence of a class of inexact interior-point algorithms for linear programs*, Mathematics of Operations Research, (to appear).

[15] D. Fulkerson, *An out-of-kilter method for minimal cost flow problems*, J. of the SIAM, 9 (1961), pp. 18–27.

[16] D. Gay, *Stopping tests that compute optimal solutions for interior-point linear programming algorithms*, tech. rep., AT&T Bell Laboratories, Murray Hill, NJ, 1989.

[17] F. Glover and D. Klingman, *The simplex SON algorithm for LP/embedded network problems*, Mathematical Programming Study, 15 (1981), pp. 148–176.

[18] A. Goldberg and R. Tarjan, *A new max-flow algorithm*, Journal of the ACM, 35 (1988), pp. 921–940.

[19] A. V. Goldberg, *An efficient implementation of a scaling minimum-cost flow algorithm*, Journal of Algorithms, (1997), pp. 1–29.

[20] D. Goldfarb and M. Grigoriadis, *A computational comparison of the Dinic and network*

simplex methods for maximum flow, Annals of Operations Research, 13 (1988), pp. 83–123.

[21] M. Grigoriadis, *An efficient implementation of the network simplex method*, Mathematical Programming Study, 26 (1986), pp. 83–111.

[22] O. Güler and Y. Ye, *Convergence behavior of interior-point algorithms*, Mathematical Programming, 60 (1993), pp. 215–228.

[23] D. Johnson and C. McGeoch, eds., *Network flows and matching: First DIMACS implementation challenge*, vol. 12 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1993.

[24] E. Johnson, *Programming in networks and graphs*, Tech. Rep. ORC 65-1, Operations Research Center, University of California, Berkeley, CA, 1965.

[25] A. Joshi, A. Goldstein, and P. Vaidya, *A fast implementation of a path-following algorithm for maximizing a linear function over a network polytope*, in Network Flows and Matching: First DIMACS Implementation Challenge, D. S. Johnson and C. C. McGeoch, eds., vol. 12 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1993, pp. 267–298.

[26] J. Kaliski and Y. Ye, *A decomposition variant of the potential reduction algorithm for linear programming*, Management Science, 39 (1993), pp. 757–776.

[27] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[28] N. Karmarkar and K. Ramakrishnan, *Computational results of an interior point algorithm for large scale linear programming*, Mathematical Programming, 52 (1991), pp. 555–586.

[29] J. Kennington and R. Helgason, *Algorithms for network programming*, John Wiley and Sons, New York, NY, 1980.

[30] M. Kojima, N. Megiddo, and S. Mizuno, *A primal-dual infeasible-interior-point algorithm for linear programming*, Mathematical Programming, 61 (1993), pp. 263–280.

[31] M. Kojima, S. Mizuno, and A. Yoshise, *A primal-dual interior point algorithm for linear programming*, in Progress in Mathematical Programming, Interior Point and Related Methods, N. Megiddo, ed., Springer, 1989, pp. 29–47.

[32] J. Kruskal, *On the shortest spanning tree of graph and the traveling salesman problem*, Proceedings of the American Mathematical Society, 7 (1956), pp. 48–50.

[33] I. Lustig, R. Marsten, and D. Shanno, *Interior point methods for linear programming: Computational state of the art*, ORSA Journal on Computing, 6 (1994), pp. 1–14.

[34] ———, *Last word on interior point methods for linear programming – For now*, ORSA Journal on Computing, 6 (1994), pp. 35–36.

[35] K. McShane, C. Monma, and D. Shanno, *An implementation of a primal-dual interior point method for linear programming*, ORSA Journal on Computing, 1 (1989), pp. 70–83.

[36] N. Megiddo, *Pathways to the optimal set in linear programming*, in Progress in Mathematical Programming, Interior Point and Related Methods, N. Megiddo, ed., Springer, 1989, pp. 131–158.

[37] S. Mehrotra, *On the implementation of a (primal-dual) interior point method*, TR 90-03, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, 1990.

[38] S. Mehrotra and J. Wang, *Conjugate gradient based implementation of interior point methods for network flow problems*, in Linear and Nonlinear Conjugate Gradient Related Methods, L. Adams and J. Nazareth, eds., SIAM, 1995.

[39] S. Mehrotra and Y. Ye, *On finding the optimal facet of linear programs*, Mathematical Programming, (1993), pp. 497–516.

[40] J. Meijerink and H. van der Vorst, *An iterative method for linear equation systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148–162.

[41] S. Mizuno and F. Jarre, *Global and polynomial time convergence of an infeasible interior point algorithm using inexact computation*, Research Memorandum 605, Institute of Statistical Mathematics, Tokyo, Japan, 1996.

[42] R. Monteiro and I. Adler, *Interior path following primal-dual algorithms. Part i: Linear programming*, Mathematical Programming, 44 (1989), pp. 27–41.

[43] R. Monteiro, T. Tsuchiya, and Y. Wang, *A simplified global convergence proof of the affine scaling algorithm*, Annals of Operations Research, 47 (1993), pp. 443–482.

[44] J. Mulvey, *Testing of a large scale network optimization program*, Mathematical Programming, 15 (1978), pp. 291–315.

[45] L. Portugal, F. Bastos, J. Júdice, J. Paixão, and T. Terlaky, *An investigation of interior point algorithms for the linear transportation problem*, SIAM J. Sci. Computing, 17 (1996), pp. 1202–1223.

[46] L. Portugal, M. Resende, G. Veiga, and J. Judice, *Preconditioners for interior point network flow methods*, tech. rep., AT&T Labs Research, Florham Park, NJ USA, 1998.

[47] R. Prim, *Shortest connection networks and some generalizations*, Bell System Technical Journal, 36 (1957), pp. 1389–1401.

[48] A. Rajan, *An empirical comparison of KORBX against RELAXT, a special code for network flow problems*, tech. rep., AT&T Bell Laboratories, Holmdel, NJ, 1989.

[49] K. Ramakrishnan, N. Karmarkar, and A. Kamath, *An approximate dual projective algorithm for solving assignment problems*, in Network Flows and Matching: First DIMACS Implementation Challenge, D. S. Johnson and C. C. McGeoch, eds., vol. 12 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1993, pp. 431–451.

[50] M. Resende, T. Tsuchiya, and G. Veiga, *Identifying the optimal face of a network linear program with a globally convergent interior point method*, in Large scale optimization: State of the art, W. Hager, D. Hearn, and P. Pardalos, eds., Kluwer Academic Publishers, 1994, pp. 362–387.

[51] M. Resende and G. Veiga, *Computing the projection in an interior point algorithm: An experimental comparison*, Investigación Operativa, 3 (1993), pp. 81–92.

[52] ———, *An efficient implementation of a network interior point method*, in Network Flows and Matching: First DIMACS Implementation Challenge, D. S. Johnson and C. C. McGeoch, eds., vol. 12 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1993, pp. 299–348.

[53] ———, *An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks*, SIAM Journal on Optimization, 3 (1993), pp. 516–537.

[54] M. G. Resende and P. M. Pardalos, *Interior point algorithms for network flow problems*, in Advances in linear and integer programming, J. Beasley, ed., Oxford University Press, 1996, pp. 147–187.

[55] B. Simeone, P. Toth, G. Gallo, F. Maffioli, and S. Pallottino, eds., *Fortran codes for network optimization*, vol. 13 of Annals of Operations Research, J.C. Baltzer, Basel, Switzerland, 1988.

[56] K. Tanabe, *Centered Newton method for mathematical programming*, in System modeling and optimization, M. Iri and K. Yajima, eds., Springer, 1988, pp. 197–206.

[57] R. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[58] T. Tsuchiya and M. Muramatsu, *Global convergence of the long-step affine scaling algorithm for degenerate linear programming problems*, SIAM Journal on Optimization, 5 (1995), pp. 525–551.

[59] P. Vaidya, *An algorithm for linear programming which requires* $o(((m+n)n^2+(m+n)^{1.5}n)l)$ *arithmetic operations*, Mathematical Programming, 47 (1990), pp. 175–201.

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

[61] Q.-J. Yeh, *A reduced dual affine scaling algorithm for solving assignment and transportation problems*, PhD thesis, Columbia University, New York, NY, 1989.