A specialized interior-point algorithm for huge minimum convex cost flows in bipartite networks

Jordi Castro
Dept. of Stat. and Oper. Res.
Universitat Politècnica de Catalunya
Barcelona, Catalonia
jordi.castro@upc.edu

Stefano Nasini
Dept. of Econ. and Quant. Meth.
IESEG School of Management
Lille, France
snasini@ieseg.fr

Research Report UPC-DEIO DR 2018-XX
November 2018

Report available from http://www-eio.upc.es/~jcastro

A specialized interior-point algorithm for huge minimum convex cost flows in bipartite networks

Jordi Castro · **Stefano Nasini**

Abstract The computation of the Newton direction is the most time consuming step of interior-point methods. This direction was efficiently computed by a combination of Cholesky factorizations and conjugate gradients in a specialized interior-point method for block-angular structured problems. In this work we apply this algorithmic approach to solve very large instances of minimum cost flows problems in bipartite networks, for convex objective functions with diagonal Hessians (i.e., either linear, quadratic or separable nonlinear objectives). After analyzing the theoretical properties of the interior-point method for this kind of problems, we provide extensive computational experiments with linear and quadratic instances of up to one billion arcs and 200 and five million nodes in each subset of the node partition. For linear and quadratic instances our approach is compared with the barriers algorithms of CPLEX (both standard path-following and homogeneous-self-dual); for linear instances it is also compared with the different algorithms of the state-of-the-art network flow solver LEMON (namely: network simplex, capacity scaling, cost scaling and cycle canceling). The specialized interior-point approach significantly outperformed the other approaches in most of the linear and quadratic transportation instances tested. In particular, it always provided a solution within the time limit and it never exhausted the 192 Gigabytes of memory of the server used for the runs. For assignment problems the network algorithms in LEMON were the most efficient option.

Keywords Interior-point methods · Minimum cost flow problems · Preconditioned conjugate gradient · Large-scale optimization

Jordi Castro
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Barcelona, Catalonia.
E-mail: jordi.castro@upc.edu

Stefano Nasini
Dept. of Economics and Quantitative Methods
IESEG School of Management, (LEM CNRS 9221)
Lille/Paris, France
E-mail: s.nasini@ieseg.fr

Mathematics Subject Classification (2000) 90C06 · 05C21 · 90C51 · 47N10

1 Introduction

Given a graph (or network) $G = (V, E)$, where V is a set of nodes (or vertices) and $E \subseteq V \times V$ is a set of edges (or arcs), the minimum cost flow problem is a widely studied optimization problem, consisting in selecting a collection of edges in E which satisfies the circulation of a certain amount of flow between specified vertices while minimizing a cost function (defined on the circulating flow). These problems encompass a broad class of applications, ranging from best delivery route [29], to statistical clustering [21], and assignment [6], among others; an extensive list of applications can be found in the reference book [1] and the survey [2]. The minimum cost flow problem holds a central position among network optimization models, as it can be solved extremely efficiently, based on specialized combinatorial algorithms, such as the cycle canceling [19], cost scaling [16], capacity scaling [15] and the network simplex [26].

An interesting subclass of minimum cost flow problems covers the special case of bipartite graphs (or bipartite networks) [1]. A graph G is bipartite if we can partition V into two subsets I and J, which we call *layers* (they might be thought as pairs of *origins–destinations*, or *agents–tasks*), so that for each arc $(i, j) \in E$, either $i \in I$ and $j \in J$, or $j \in I$ and $i \in J$. A particular type of minimum cost flows in bipartite networks is the assignment—or weighted matching—problem [6], where $|I| = |J|$, supplies and demands are $+1$ and -1 for, respectively, nodes in I and J, and arc capacities are 1.

A natural way to enlarge the range of applicability of this class of problems is to allow for more general cost structures, such as the case of concave or convex objective functions. As noted by [29] more than half a century ago, *"the literature is replete with analyses of minimum cost flows in networks for which the cost of shipping from node to node is a linear function. However, the linear cost assumption is often not realistic. Situations in which there is a set-up charge, discounting, or efficiencies of scale give rise to concave functions"*. Contextually, strict convexity arises when shipments or production respectively encounter congestion or inefficiencies of scale.

Although the problem of finding a minimum cost matching is as easy as matrix inversion (see [22]), the minimum convex cost flows in bipartite networks is not (see [23] for a comprehensive experimental study on testing large-scale minimum cost network flow problems). From the algorithmic viewpoint, the presence of nonlinear costs prevents the use of the aforementioned specialized algorithms for classical minimum cost flow problems (see chapter 14 of [1]), opening the possibility of an efficient application of specialized interior-point methods (IPMs from now on).

The main claim of this paper is that minimum convex cost flows problems in bipartite networks (MCCFBN from now on) with diagonal Hessians (i.e., either linear, quadratic or separable nonlinear objectives) exhibit particularly favorable primal-block angular structures, which facilitate the efficient application of the specialized IPM for primal-block angular problems of [8, 9, 10]. The underlying idea of this algorithmic approach consists in solving the normal equations (associated to the Newton direction of the IPM) by a combination of Cholesky factorizations for the block con-

straints and preconditioned conjugate gradient (PCG from now on) iterations for the linking constraints.

From the theoretical and the empirical viewpoint, this work shows that minimum cost flows problems in bipartite networks can be included within the successful application of the specialized IPM for primal block-angular problems. After uncovering a collection of theoretical properties on the asymptotic behavior of the inner PCG procedure for MCCFBN problems, we provide extensive computational experiments with linear and quadratic instances of up to one billion arcs and 200 and five million nodes in each subset of the node partition. This is in line with the the growing effort to address the design of scalable algorithmic solutions for large-scale optimization [5, 9, 12, 13]. In fact, the presented results provide both a theoretical and computational support of the scalable property of the the specialized IPM for MCCFBN problems.

For linear and quadratic instances our approach is compared with the barriers algorithms of CPLEX (both standard path-following and homogeneous-self-dual); for linear instances it is also compared with the different algorithms (namely: network simplex, capacity scaling, cost scaling and cycle canceling) of the state-of-the-art network flow solver LEMON [20]. The specialized IPM significantly outperformed the other approaches in most of the linear and quadratic transportation instances tested. In particular, it always provided a solution within the time limit and it never exhausted the 192 Gigabytes of memory of the server used for the runs. However, for the (linear) assignment problems tested, one of the LEMON algorithms was always the most efficient choice, as it will be discussed in the section of computational results.

The rest of this paper is organized as follows. In Section 2 the block-angular formulation of the MCCFBN problem is introduced. The design of the specialized IPM for the MCCFBN problem is presented in Section 3. Section 4 contains theoretical properties on the asymptotic behavior of the specialized IPM for different parameter configurations of the MCCFBN problem. An extensive computational analysis is carried out in Section 5. The paper ends with Section 6 highlighting the main findings in this work.

2 Problem description and formulation

The MCCFBN problem is defined in a network with a set of nodes J (associated to customers or tasks), with $m = |J|$, whose demands (or requirements) are known and are to be supplied from another set of nodes I (operating suppliers or machines), with $n = |I|$. We consider costs reflecting the transportation (or assignment) between each pair $(i, j) \in I \times J$. The goal is to decide how to supply the customers (or how to carry out the tasks) from the operating suppliers (or by the operating machines) in order to minimize the aggregate transportation/assignment cost. Before presenting an optimization model for this problem we introduce some notation that will be considered hereafter:

- $f_{ij} : \mathbb{R} \to \mathbb{R}$, convex cost function of flow from $i \in I$ to $j \in J$;
- $d_j \in \mathbb{R}_+$, demand of $j \in J$;
- $s_i \in \mathbb{R}_+$, supply (or supply capacity) of $i \in I$;

– $u_{ij} \in \mathbb{R}_+$, capacity of the arc $(i,j) \in I \times J$;

where $\mathbb{R}$ and $\mathbb{R}_+$ are the sets of real and nonnegative real numbers respectively. The decisions to be made are the flows from $i \in I$ to $j \in J$, which we denote as x_{ij}. The MCCFBN problem can be formulated as follows:

$$\min \quad \sum_{i\in I}\sum_{j\in J} f_{ij}(x_{ij}), \tag{1}$$

$$\text{subject to} \quad \sum_{i\in I} x_{ij} = d_j, \qquad j \in J, \tag{2}$$

$$\sum_{j\in J} x_{ij} \le s_i \qquad i \in I, \tag{3}$$

$$0 \le x_{ij} \le u_{ij}, \quad i \in I,\ j \in J. \tag{4}$$

When u_{ij} ($i \in I,\ j \in J$) are sufficiently large, feasibility is guaranteed as long as $\sum_{j\in J} d_j \le \sum_{i\in I} s_i$. If $\sum_{j\in J} d_j = \sum_{i\in I} s_i$, then s_i are supplies (instead of supply capacities), and constraints (3) are active in a solution. Problem (1)-(4) can be rewritten in a block-angular form:

$$\min \quad \mathbf{f}(\mathbf{x}) \triangleq \sum_{j\in J} \mathbf{f}_j(\mathbf{x}_j) \tag{5}$$

$$\text{subject to} \quad \begin{bmatrix} \mathbf{e}^\top & & & \\ & \mathbf{e}^\top & & \\ & & \ddots & \\ & & & \mathbf{e}^\top \\ \mathbb{I} & \mathbb{I} & \dots & \mathbb{I} \ \ \mathbb{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_m \\ \mathbf{x}_0 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \\ \mathbf{s} \end{bmatrix} \tag{6}$$

$$0 \le \mathbf{x}_j \le \mathbf{u}_j, \quad j = 0,1,\dots,m, \tag{7}$$

where $\mathbf{x}_j = [x_{1j},\dots,x_{nj}]^\top \in \mathbb{R}^n$ represents the flows sent from I to node j; $\mathbf{f}_j(\mathbf{x}_j) = \sum_{i\in I} f_{ij}(x_{ij})$ is the cost of flows arriving in node j; $\mathbf{x} = [\mathbf{x}_1^\top,\dots,\mathbf{x}_m^\top]^\top \in \mathbb{R}^{mn}$ is the vector of flows; $\mathbf{f}(\mathbf{x})$ is the objective function; matrix $\mathbb{I} \in \mathbb{R}^{n\times n}$ is the identity matrix; $\mathbf{e} \in \mathbb{R}^n$ is a vector of ones; $\mathbf{x}_0 \in \mathbb{R}^n$ is the vector of slacks of the linking constraints, which represents the unused supply capacity; and $\mathbf{s} = [s_1,\dots,s_n]^\top \in \mathbb{R}^n$ is the right-hand side vector for the linking constraints, containing all the supplies. Note that the m block constraints $\mathbf{e}^\top \mathbf{x}_j = d_j$ correspond to (2), whereas the linking constraints $\sum_{j\in J} \mathbb{I}\mathbf{x}_j + \mathbf{x}_0 = \mathbf{s}$ correspond to (3). Note that this reformulation and matrix structure is also valid for generalized flows (see chapter 15 of [1]), just by replacing $\mathbf{e}^\top$ by some vector and $\mathbb{I}$ by a diagonal matrix.

3 Outline of the specialized IPM for primal block-angular problems

Formulation (5)–(7) exhibits a primal block-angular structure, and thus it can solved by the interior-point method of [8, 10, 11]. This method is a specialized primal-dual path-following algorithm tailored for primal block-angular problems. A thorough description of primal-dual path-following algorithms can be found in [24] and [28].

Let $A \in \mathbb{R}^{(m+n)\times(nm+n)}$ and $\mathbf{b} \in \mathbb{R}^{m+n}$ denote respectively the coefficient matrix and right-hand-side of (5)–(7); $\mathbf{z} \in \mathbb{R}^{mn}$ and $\mathbf{w} \in \mathbb{R}^{mn}$ the vector of Lagrange multipliers for the lower and upper bounds (7); $\nabla\mathbf{f}(\mathbf{x}) \in \mathbb{R}^{mn}$ the gradient vector of the objective function; and $\lambda \in \mathbb{R}^{m+n}$ the vector of Lagrange multipliers of the equality constraints (6). For any $\mu \in \mathbb{R}_+$ the central path can be derived as a solution of the μ-perturbed Karush-Kuhn-Tucker optimality conditions of (5)–(7):

$$\begin{aligned}
\mathbf{r}_b &\equiv \mathbf{b} - A\mathbf{x} &= 0, &(8)\\
\mathbf{r}_f &\equiv \nabla\mathbf{f}(\mathbf{x}) - A^\top\lambda - \mathbf{z} + \mathbf{w} &= 0, &(9)\\
\mathbf{r}_{xz} &\equiv \mu\mathbf{e} - XZ\mathbf{e} &= 0, &(10)\\
\mathbf{r}_{xw} &\equiv \mu\mathbf{e} - (U-X)W &= 0, &(11)\\
&\quad (\mathbf{x}, \mathbf{z}, \mathbf{w}) &\geq 0, &(12)
\end{aligned}$$

where X, Z and W are diagonal matrices whose diagonal entries are those of $\mathbf{x}$, $\mathbf{z}$ and $\mathbf{w}$ respectively. Matrix U is a diagonal matrix whose diagonal entries are the upper limits $\mathbf{u}_1 \ldots \mathbf{u}_m$. The primal-dual path-following method consists in solving the nonlinear system (8)–(12) by a sequence of damped Newton's directions (with step-length reduction to preserve the nonnegativity of variables), decreasing the value of μ at each iteration. On the left-hand side of (8)–(12) we have explicitly reported the residuals of the current iterate $\mathbf{r}_b$, $\mathbf{r}_f$, $\mathbf{r}_{xz}$ and $\mathbf{r}_{xw}$.

The Newton's iterates are carried out by forming a linear system of variables $\Delta\mathbf{x}$, $\Delta\lambda$, $\Delta\mathbf{z}$, and $\Delta\mathbf{w}$ around the current iterate—i.e. a linear approximation of (8)–(12) around the current point. By letting $\nabla^2\mathbf{f}(\mathbf{x})$ be the Hessian of $\mathbf{f}(\mathbf{x})$ at the current iterate, with few algebraic operations (see, for instance, [28] for details) we can obtain the Newton direction in the dual space $\Delta\lambda$ by solving the system of normal equations

$$(A\Theta A^\top)\Delta\lambda = \mathbf{g}, \tag{13}$$

where

$$\begin{aligned}
\mathbf{g} &= \mathbf{r}_b + A\Theta(\mathbf{r}_f + (U-X)^{-1}\mathbf{r}_{xw} - X^{-1}\mathbf{r}_{xz}) \ \in \ \mathbb{R}^{n+m},\\
\Theta &= (ZX^{-1} + W(U-X)^{-1} + \nabla^2\mathbf{f}(\mathbf{x}))^{-1} \ \in \ \mathbb{R}^{(nm)\times(nm)}.
\end{aligned} \tag{14}$$

The values of $\Delta\mathbf{x}$, $\Delta\mathbf{w}$ and $\Delta\mathbf{z}$ can be easily computed once $\Delta\lambda$ is known. Note that Θ is a diagonal matrix as long as $\mathbf{f}(\mathbf{x})$ is separable (i.e., Hessian is diagonal), which is the case for the MCCFBN problem (1)-(4).

Solving (13) is the most expensive computational step of the interior-point method [24, 28]. General interior-point solvers usually compute (13) by a Cholesky factorization, while the specialized method of [8, 10, 11] combines Cholesky with PCG. Exploiting the structure of A in (6), and appropriately partitioning Θ and $\Delta\lambda$ according to the $m+1$ blocks of variables and constraints, we have

$$A\Theta A^\top \Delta\lambda = \left[\begin{array}{ccc|c} \mathbf{e}^\top\Theta_1\mathbf{e} & & & \mathbf{e}^\top\Theta_1 \\ & \ddots & & \vdots \\ & & \mathbf{e}^\top\Theta_m\mathbf{e} & \mathbf{e}^\top\Theta_m \\ \hline \Theta_1\mathbf{e} & \dots & \Theta_m\mathbf{e} & \Theta_0+\sum_{j=1}^m\Theta_j \end{array}\right]\Delta\lambda$$

$$= \left[\begin{array}{ccc|c} \mathrm{Tr}(\Theta_1) & & & \varphi_1^\top \\ & \ddots & & \vdots \\ & & \mathrm{Tr}(\Theta_m) & \varphi_m^\top \\ \hline \varphi_1 & \dots & \varphi_m & D \end{array}\right]\left[\begin{array}{c} \Delta\lambda_{1_1} \\ \vdots \\ \Delta\lambda_{1_m} \\ \hline \Delta\lambda_2 \end{array}\right] = \begin{bmatrix} B & C \\ C^\top & D \end{bmatrix}\begin{bmatrix}\Delta\lambda_1 \\ \Delta\lambda_2\end{bmatrix} = \begin{bmatrix}\mathbf{g}_1 \\ \mathbf{g}_2\end{bmatrix}, \tag{15}$$

where $\mathrm{Tr}(.)$ denotes the trace of a matrix, $\varphi_j = [\Theta_{j_{11}},\dots,\Theta_{j_{nn}}]^\top$, for $j = 1,\dots,m$, and $D = \Theta_0 + \sum_{j=1}^m \Theta_j$ is a diagonal matrix. By eliminating $\Delta\lambda_1$ from the first group of equations, the system (15) reduces to

$$\left(D - C^\top B^{-1} C\right)\Delta\lambda_2 = (g_2 - C^\top B^{-1} g_1) \tag{16}$$

$$B\Delta\lambda_1 = (g_1 - C\Delta\lambda_2). \tag{17}$$

Since B is diagonal, system (17) is directly solved as $\Delta\lambda_{1,j} = (g_1 - C\Delta\lambda_2)_j / \mathrm{Tr}(\Theta_j)$, for $j = 1\dots m$. The only computational effort is thus the solution of system (16)—the Schur complement of (15)—whose dimension is n, the number of nodes in I. This might be computationally expensive if solved by Cholesky factorization, as it requires the computation of a potentially dense matrix $D - C^\top B^{-1} C$. By contrast, the use of PCG was suggested in [8], using a preconditioner based on the following power series expansion of the inverse of the Schur complement (see [8, Prop. 4] for a proof):

$$(D - C^\top B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty}\left(D^{-1}(C^\top B^{-1} C)\right)^i\right) D^{-1}. \tag{18}$$

The preconditioner is obtained by taking a finite number of terms of the infinite power series (18). In this paper we consider the truncation at the first term, which implies the definition of D^{-1} as a preconditioner. In such case, the solution of (16) by the conjugate gradient only requires matrix-vector products with matrix $(D - C^\top B^{-1} C)$ —which is computationally cheap because of the structure of D, C and B— and the solution of systems with matrix D —which are straightforward since D is diagonal. It is worth mentioning that the computation of approximate Newton directions by PCG does not preclude the convergence of IPMs, as shown in [17]. There is an extensive literature on the use of PCG within IPMs for different types of problems (see for instance, [3, 4, 7, 14, 25, 27]). An alternative preconditioner to the power series one was suggested in [11].

The quality of the preconditioner depends on the spectral radius (i.e., the maximum absolute eigenvalue) of matrix $D^{-1}(C^\top B^{-1} C)$, which is real and always in $[0,1)$, as shown in [10] (the closest to zero, the better is the preconditioner). However, in the last interior-point iterations matrix Θ becomes very ill-conditioned (i.e.

some of its elements go to zero whereas others tend to infinity), so that the spectral radius of matrix $D^{-1}(C^\top B^{-1}C)$ approaches the one, degrading the performance of the conjugate gradient and hindering the efficient solution of (13) (see [10] for more details).

For a general problem, the spectral radius can not be (efficiently) computed, but it can be upper bounded using Theorem 1 in [10]. For the particular case of minimum convex cost flow in bipartite networks, this upper bound can be replaced by the equality provided by next lemma:

Lemma 1 (From Theorem 1 in [10]) *Let ρ be the spectral radius of matrix $D^{-1}(C^\top B^{-1}C)$ and $\mathbf{v}$ the eigenvector (or one of the eigenvectors) of $D^{-1}(C^\top B^{-1}C)$ for ρ. Then we have that*

$$\rho = \frac{\kappa}{\mathbf{v}^\top \left(\Theta_0 + \sum_{i=1}^{m} \Theta_i\right) \mathbf{v}} \quad \text{where} \quad \kappa = \mathbf{v}^\top \left(\sum_{i=1}^{m} \Theta_i \mathbf{e} \left(\mathbf{e}^\top \Theta_i \mathbf{e} \right)^{-1} \mathbf{e}^\top \Theta_i \right) \mathbf{v}. \tag{19}$$

In the next section, asymptotic cases for the computation of $\Delta\lambda_2$ are studied, along with the behaviour of ρ under different parameterizations of (16) (i.e. dimensions of the network layers and relationship between the total demand and the total supply).

4 Asymptotic behavior of the specialized IPM for MCCFBN problems

To make a compelling case of the scalable properties of the specialized IPM for MCCFBN problems, as announced in Section 1, asymptotic behaviors boosting the computation of (16) at each IPM iteration are studied hereafter.

From (1)-(4) we see that the number of constraints and variables grows as $m+n$ and $nm+n$, respectively, and that the system (16) of dimension n has to be solved at each IPM iteration. Thus, we expect the number of nodes in I (the suppliers) to have a strong negative impact on the computational efficiency of the proposed approach (as well as for other network flow algorithms, whose computational performances rely on the problem size).

To counterbalance this size effect, the next proposition shows that, asymptotically, the power series preconditioner D^{-1} provides the inverse of the matrix in the Schur complement system (16) when n grows larger.

Proposition 1 *Let us assume that there is an $\varepsilon > 0$ such that the current interior-point $(\mathbf{x},\mathbf{z},\mathbf{w})$ satisfies $\mathbf{u} - \varepsilon > \mathbf{x} > \varepsilon$, $\mathbf{z} > \varepsilon$, $\mathbf{w} > \varepsilon$ (which is known to be the case in all IPM iterations [28]). Then, the entries of $C^\top B^{-1}C$ are $O(n^{-1})$, and then when $n \to \infty$ we have $C^\top B^{-1}C \to 0$.*

Proof From the definition of C and B in (15), since B is diagonal, we have that entry (h,l) of $C^\top B^{-1}C$ is

$$(C^\top B^{-1}C)_{hl} = \sum_{j=1}^{m} \frac{\Theta_{j,hh}\Theta_{j,ll}}{\sum_{i=1}^{n} \Theta_{j,ii}} \leq \frac{1}{n} \sum_{j=1}^{m} \frac{\Theta_{j,hh}\Theta_{j,ll}}{\min_i \Theta_{j,ii}}. \tag{20}$$

Since $\Theta = (ZX^{-1} + W(U-X)^{-1} + \nabla^2\mathbf{f}(\mathbf{x}))^{-1}$ and $(\mathbf{x},\mathbf{z},\mathbf{w}) > \varepsilon$ and $\mathbf{u}-\mathbf{x} > \varepsilon$ we have from (20) that $(C^\top B^{-1}C)_{hl} = O(n^{-1})$, and

$$\lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{m} \frac{\Theta_{j,hh}\Theta_{j,ll}}{\min_i \Theta_{j,ii}} = 0.$$

□

Proposition 1 guarantees that $D - C^\top B^{-1}C \to D$ when $n \to \infty$, and then D^{-1} is the best preconditioner. Under this asymptotic case the entire dual direction can also be analytically computed.

$$\Delta\lambda_{2,i} = \frac{(g_2 - C^\top B^{-1} g_1)_i}{\Theta_{0,ii} + \sum_{j=1}^m \Theta_{j,ii}}, \qquad \text{for } i = 1,\ldots,n,$$
$$\Delta\lambda_{1,j} = \frac{(g_1 - C\Delta\lambda_2)_j}{\mathrm{Tr}(\Theta_j)} \qquad \text{for } j = 1,\ldots,m.$$

As a result of Lemma 1 and Proposition 1, and noting that the total slack in the defined MCCFBN is fixed by construction at the excess capacity (i.e. $\sum_{i=1}^n x_{0,i} = \sum_{i=1}^n s_i - \sum_{j=1}^m d_j$), we obtain a direct relationship between the spectral radius ρ, the number of suppliers n, the total excess capacity, and the duality gap μ at the current IPM iteration:

Proposition 2 (Dependency of the spectral radius) *Let $\hat{s} = \sum_{i=1}^n s_i$, $\hat{d} = \sum_{j=1}^m d_j$. Under the hypotheses of Lemma 1 and Proposition 1, at each point of the central path (i.e., a point solving the μ-perturbed Karush-Kuhn-Tucker conditions), we have*

$$\rho = \frac{O\left(n^{-1}\right)}{\frac{1}{\mu} O\left((\hat{s}-\hat{d})^2\right) + \gamma} \quad \textit{where} \quad \gamma = \mathbf{v}^\top \left(\sum_{j=1}^m \Theta_j\right) \mathbf{v} \geq 0. \tag{21}$$

Proof Since $\mathbf{v}$ is one of the orthonormal eigenvectors of $D^{-1}(C^\top B^{-1}C)$, we have that $\|\mathbf{v}\| = 1$. From Proposition 1, all the entries of $(C^\top B^{-1}C)$ are $O(n^{-1})$. It follows that the term $\kappa = \mathbf{v}^\top (C^\top B^{-1}C)\mathbf{v}$ of (19) is $O(n^{-1})$.

Since $u_{0,i} = \infty$ (MCCFBN problems have no upper bound on the excess capacity) and $\mathbf{f}_0(\mathbf{x}_0) = 0$ (the excess capacity has no cost), from (14) $\Theta_{0,ii} = x_{0,i}/z_{0,i} = x_{0,i}^2/\mu$, using for the last equality that the point is on the central path (that is, $x_{0_i} z_{0,i} = \mu$). Then $\mathbf{v}^\top \Theta_0 \mathbf{v}^\top = \frac{1}{\mu}\sum_{i=1}^n v_i^2 x_{0,i}^2 = \frac{1}{\mu} O\left(\max_{i\in I} x_{0,i}^2\right)$, and from (19)

$$\rho = \frac{O\left(n^{-1}\right)}{\frac{1}{\mu} O\left(\max_{i\in I} x_{0,i}^2\right) + \mathbf{v}^\top \left(\sum_{i=1}^m \Theta_i\right) \mathbf{v}} = \frac{O\left(n^{-1}\right)}{\frac{1}{\mu} O\left((\hat{s}-\hat{d})^2\right) + \mathbf{v}^\top \left(\sum_{i=1}^m \Theta_i\right) \mathbf{v}},$$

where the second equality has been obtained by noting that $\max_{i\in I} x_{0,i} \leq \sum_{i=1}^n x_{0,i} = \sum_{i=1}^n s_j - \sum_{j=1}^m d_j$, and $x_{0,i} \geq 0$ for every $i \in I$.

□

On the one hand, as a consequence of Proposition 2, the efficiency of the PCG is enhanced by the excess capacity and the number of suppliers n. (These relationships are numerically supported in Section 5). On the other hand, Proposition 2 implies that the impact of the excess capacity on the performance of the PCG becomes more and more relevant when the optimal solution is approached (i.e. when μ goes to zero). This may explain the good behaviour of the preconditioner for MCCFBN near the optimal solution of MCCFBN, unlike it was observed for other problems [8, 10, 9]. It is not surprising that the excess capacity improves the performance of the method, since it reduces the number of active linking constraints in the optimal point, thus increasing the block-separability of the problem.

5 Computational experiments

The specialized interior-point algorithm for MCCFBN described in previous sections was developed using the BlockIP solver [9], an efficient C++ implementation of the specialized IPM for general block-angular problems. This code will be denoted MCCFBN-BlockIP or simply BlockIP hereafter, and a copy of it (in binary format) can be obtained from `http://www-eio.upc.es/~jcastro/MCCFBN-BlockIP.html`. This section describes the computational experiments designed to numerically assess the performance of MCCFBN-BlockIP and its competitive advantage with respect to cutting edge solvers for this class of problems. Specifically, we considered CPLEX 12.5—a state-of-the-art general optimization package—and LEMON—a highly efficient network optimization code. LEMON was selected because according to [20] (and to some preliminary tests we carried out) it outperformed the best current network optimization packages (some of them implementing algorithms already available in LEMON, and others implementing different approaches, such as alternative specialized interior-point methods based on tree preconditioners [27]). Therefore, through LEMON and the results in [20], our algorithm can also be indirectly compared with these other network optimization packages. For the computational executions we considered the following algorithms implemented in CPLEX 12.5 and LEMON (which are the most efficient ones for MCCFBN problems):

- CPLEX (release 12.5) implementation of the barrier algorithm;
- CPLEX (release 12.5) implementation of the dual simplex algorithm;
- LEMON (release 1.3.1) implementation of the capacity scaling algorithm;
- LEMON (release 1.3.1) implementation of the cost scaling algorithm;
- LEMON (release 1.3.1) implementation of the cycle canceling algorithm;
- LEMON (release 1.3.1) implementation of the network simplex algorithm.

While the barrier algorithm can be applied both to linear and (quadratic) convex instances, the remaining solvers are only applied to linear instances. Thus, the barrier algorithm can be regarded as the current benchmark for general MCCFBN problems. For running the CPLEX barrier we considered one thread, and no crossover (otherwise the CPU time would significantly increase). Similarly, for LEMON and for MCCFBN-BlockIP a single thread was used.

The collection of computational experiments of the following subsections are grouped in three categories:

- Tests to assess the effect of demand slack on the computational performance of MCCFBN-BlockIP;
- Tests to assess the competitive advantage of MCCFBN-BlockIP when solving transportation instances: (i) with linear integer costs (since some of the LEMON algorithms only work with integer values); (ii) with linear fractional costs; (iii) with quadratic—and fractional—costs.
- Tests to assess the competitive advantage of MCCFBN-BlockIP when solving assignment instances: (i) with small costs; (ii) with large costs

Assignment instances were obtained with the DIMACS generator by McGeoch [18]. Transportation instances were obtained with a new generator which considers a spatial two-dimensional distribution of the graph, and writes the problem in DIMACS format. For convenience, the source code of both generators is also available from `http://www-eio.upc.es/~jcastro/MCCFBN-BlockIP.html`.

For the large-scale instances tested in this work the optimality tolerance for CPLEX and MCCFBN-BlockIP was set to 10^{-4}; this tolerance cannot be adjusted in LEMON, so default values were used (however, CPLEX, LEMON and MCCFBN-BlockIP provided solutions with the same objective function). All the runs were carried out on a Fujitsu Primergy RX2540 M1 4X server with two 2.6 GHz Intel Xeon E5-2690v3 CPUs (48 cores) and 192 Gigabytes of RAM, under a GNU/Linux operating system (openSuse 13.2), without exploitation of multithreading capabilities (i.e., as noted above, a single thread was used—runs were carried out sequentially). The GCC version 4.8.3 suite of compilers (C, C++, and Fortran) was used for the implementations. A time limit of 18000 CPU seconds (5 CPU hours) was considered in all the executions.

5.1 The effect of demand slack

Noting that the slack in the defined MCCFBN is fixed by construction at the excess capacity (i.e. $\sum_{i=1}^{n} x_{0,j} = \sum_{i=1}^{n} s_i - \sum_{j=1}^{m} d_j$), Proposition 2 suggests a direct relationship between the spectral radius ρ and the total excess capacity.

The impact of the relative slack on the computational performance of the specialized IPM for MCCFBN problems is studied in this section, based on computational experiments involving 12 problem instances solved by the aforementioned MCCFBN-BlockIP, CPLEX and LEMON solvers. We considered two values of n (25 and 200), two values of m (10000 and 2000000) and three values of $1-(\hat{s}-\hat{d})/\hat{s}$ (0.1, 0.5, 1.0) to build 12 transportation instances with linear costs. Note that $1-(\hat{s}-\hat{d})/\hat{s}$ is 1 when $\hat{s}=\hat{d}$ —no slack—, and it approaches 0 when $\hat{s} \gg \hat{d}$.

The contour curves in Figure 1 illustrate the impact of the term $1-(\hat{s}-\hat{d})/\hat{s}$ on the average number of PCG iterations within each IPM iteration, for different instance sizes n and m. The different values of PCG iterations are marked in different colors, as shown by the bar at the right of the plots. It is clear from Figure 1 that the complementary of the relative demand slack $1-(\hat{s}-\hat{d})/\hat{s}$ has a clear impact on the average number of PCG iterations, that is, the less the demand slack, the more PCG iterations are required by MCCFBN-BlockIP.

(a) Excess capacity $\times$ n (b) Excess capacity $\times$ m

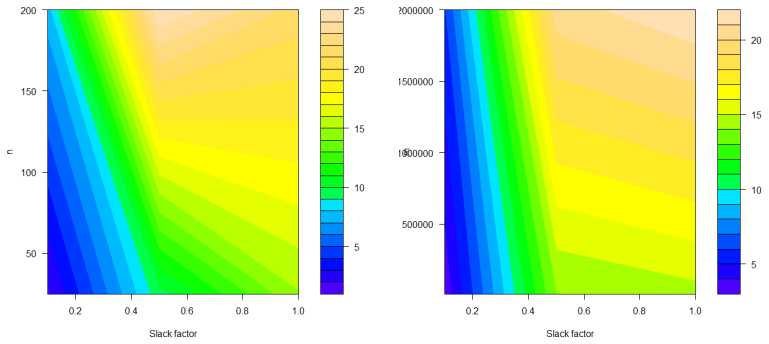

Fig. 1: The average number of PCG iterations within the IPM iterations (using different colors) is plotted for different combination of parameters. On the left panel, the axis of the contour curves report the number of nodes in the first layer n and the complementary of the relative demand slack $1-(\hat{s}-\hat{d})/\hat{s}$. On the right panel, the axis of the contour curves report the number of nodes in the second layer m and the complementary of the relative demand slack $1-(\hat{s}-\hat{d})/\hat{s}$.

5.2 Transportation instances with integer costs

We generated 20 instances with linear integer costs using the spatial generator, which takes as some of the input parameters the number of nodes in I (n) and J (m), and it builds the transportation problems by randomizing the demand points on a two-dimensional surface. These 20 instances were obtained by considering all the combinations of $n \in \{25,100,200\}$ and $m \in \{1e4,5e4,1e5,5e5,1e6\}$. In all the cases $(\hat{s}-\hat{d})/\hat{s}=0$ (that is, we consider no relative demand slack), which, according to the discussion of previous section, is the worst scenario for MCCFBN-BlockIP. These instances, being linear with integer costs, could be solved with all the available algorithms of the three solvers: the Newton and predictor-corrector interior-point algorithms of MCCFBN-BlockIP; the dual simplex and the two interior-point algorithms (that is, the standard barrier and the homogeneous-self-dual (HSD)) of CPLEX; and the capacity scaling, cost scaling, cycle canceling and network simplex of LEMON. The primal simplex of CPLEX was not considered since it was outperformed by the dual simplex.

Table 1 reports the dimension of the problems (in millions of variables and constraints), as well as the CPU time needed by the different algorithms of the three solvers. Executions marked with "—" exhausted the 18000 seconds CPU time limit. The fastest executions are marked in boldface. Note that the number of variables is $nm+n$ while the number of constraints is $n+m$; the largest instance has one billion variables and five millions constraints. From Table 1 we have that in general MCCFBN-BlockIP outperformed the other two solvers, mainly when the size of the problem is very large. To confirm that all the solvers reach points of equivalent quality, Table 2 reports the optimal objective functions associated to the problem instances in Table 1: it is seen that MCCFBN-BlockIP provides points as good as those

n.var	n.cons	BlockIP		CPLEX			LEMON			
(M)	(M)	Newton	PreCorr	HSD	Barrier	Dual S.	CapS	CosS	CycC	NetS
0.25	0.01	1.4	**1.2**	1.4	1.4	4.8	48.9	3.8	2.1	3.0
2.5	0.10	31.5	**4.0**	24.0	13.6	262.2	8817.0	330.9	104.5	1118.2
12.5	0.50	146.6	**71.4**	137.4	79.1	5265.2	—	—	2265.4	—
25	1.00	253.8	**37.4**	260.3	153.7	18071.8	—	—	10284.8	—
125	5.00	998.4	**987.9**	2267.6	1019.0	41188.5	—	—	—	—
0.5	0.01	**2.0**	2.9	3.4	2.4	11.5	82.9	6.5	3.5	2.4
5	0.10	90.9	37.1	76.9	**34.2**	1630.5	17494.0	618.1	183.5	639.3
25	0.50	518.3	237.7	478.4	**221.0**	18098.2	—	15195.3	4127.1	—
50	1.00	966.3	519.0	1295.2	**435.9**	18947.1	—	—	—	—
250	5.00	1578.2	**1497.3**	6245.7	1980.5	18043.7	—	—	—	—
1	0.01	8.4	7.4	6.6	6.5	35.8	209.2	13.1	12.7	**4.0**
10	0.10	131.3	86.1	157.2	**82.9**	6814.0	—	—	586.9	803.1
50	0.50	930.5	519.6	2424.4	536.5	5468.9	—	**462.0**	6024.3	—
100	1.00	2631.4	**771.9**	4285.4	851.2	18017.8	—	—	—	—
500	5.00	**3924.5**	5979.6	—	—	—	—	—	—	—
2	0.01	31.2	28.2	26.4	16.0	309.2	560.9	25.6	29.6	**4.2**
20	0.10	389.4	693.2	1402.0	**268.1**	18114.9	—	—	774.0	1737.3
100	0.50	2194.7	**1116.1**	7209.0	1138.7	14018.4	—	—	3510.9	—
200	1.00	3629.8	**1411.6**	15573.3	2300.9	22516.3	—	—	8260.8	—
1000	5.00	**2340.4**	2706.9	—	—	—	—	4599.1	4181.2	—

— No solution found within the time limit

Table 1: Computational performance for transportation instances with integer costs; fastest execution in boldface. The first two columns describe the problem size in terms of the number of variables ($nm+n$) and constraints ($m+n$), while the remaining columns report the CPU time of the different algorithms of MCCFBN-BlockIP, CPLEX and LEMON solvers.

computed by CPLEX and LEMON. Finally, Table 3 casts a closer look at the internal dynamics of the interior-point algorithms of MCCFBN-BlockIP (Newton and predictor-corrector) and CPLEX (standard barrier and HSD). For the four algorithms the table provides the CPU time, objective function and number of IPM iterations; for the two MCCFBN-BlockIP variants it also reports the overall number of PCG iterations. The best CPU times and objective functions are marked in boldface. Due to the use of a PCG iterative solver, the systems solved by MCCFBN-BlockIP are inexact, and thus it takes a larger amount of IPM iterations than CPLEX; however it remains globally more efficient. It is worth noting that the predictor-corrector direction is, in general, the most efficient alternative, despite it means two solutions by a PCG. We also remark that the largest instances (of 500M and 1000M variables) could not be solved by CPLEX within the five hours time limit, while MCCFBN-BlockIP solved them in less than one hour.

5.3 Transportation instances with fractional costs

The set of computational tests presented in this section mirrors the one already presented in Subsection 5.2, with the only difference given by the use of linear fractional costs. For LEMON, only network simplex is used, as according to LEMON manual, this is the only LEMON algorithm that may work with fractional costs. In many instances, however, it failed, as shown in Tables 4 and 5.

From Table 4, the use of fractional costs enhances the competitive advantage of MCCFBN-BlockIP over CPLEX and LEMON: for all the instances MCCFBN-BlockIP reported the fastest execution; and, unlike for the case of integer costs, the

n.var	n.cons	BlockIP		CPLEX			LEMON			
(M)	(M)	Newton	PreCorr	HSD	Barrier	Dual S.	CapS	CosS	CycC	NetS
0.25	0.01	**6.284e+11**	6.285e+11	6.288e+11	**6.284e+11**	**6.284e+11**	**6.284e+11**	**6.284e+11**	**6.284e+11**	**6.284e+11**
2.5	0.10	**6.27e+13**	6.536e+13	6.28e+13	6.271e+13	6.271e+13	6.271e+13	6.271e+13	6.271e+13	6.271e+13
12.5	0.50	**1.567e+15**	**1.567e+15**	1.57e+15	**1.567e+15**	**1.567e+15**	—	—	**1.567e+15**	—
25	1.00	**6.268e+15**	6.687e+15	6.278e+15	6.269e+15	—	—	—	6.269e+15	—
125	5.00	**3.499e+16**	**3.499e+16**	3.501e+16	3.512e+16	—	—	—	—	—
0.5	0.01	**4.401e+12**	4.402e+12	4.402e+12	4.402e+12	4.402e+12	4.402e+12	4.402e+12	4.402e+12	4.402e+12
5	0.10	**4.382e+14**	**4.382e+14**	4.388e+14	**4.382e+14**	**4.382e+14**	**4.382e+14**	**4.382e+14**	**4.382e+14**	**4.382e+14**
25	0.50	**9.91e+15**	1.095e+16	1.097e+16	1.095e+16	—	—	1.095e+16	1.095e+16	—
50	1.00	**2.438e+16**	2.56e+16	2.564e+16	2.56e+16	—	—	—	—	—
250	5.00	**1.828e+16**	**1.828e+16**	1.829e+16	**1.828e+16**	—	—	—	—	—
1	0.01	**3.643e+13**	**3.643e+13**	3.644e+13	**3.643e+13**	**3.643e+13**	**3.643e+13**	**3.643e+13**	**3.643e+13**	**3.643e+13**
10	0.10	**3.61e+15**	3.611e+15	3.617e+15	3.611e+15	3.611e+15	—	—	3.611e+15	3.611e+15
50	0.50	**3.77e+15**	4.898e+15	4.899e+15	4.9e+15	4.898e+15	—	4.898e+15	4.898e+15	—
100	1.00	**6.854e+15**	6.869e+15	6.871e+15	6.869e+15	—	—	—	—	—
500	5.00	**1.55e+16**	**1.55e+16**	—	—	—	—	—	—	—
2	0.01	2.9e+14	2.9e+14	2.9e+14	2.9e+14	**2.899e+14**	**2.899e+14**	**2.899e+14**	**2.899e+14**	**2.899e+14**
20	0.10	**3.028e+15**	3.036e+15	3.038e+15	3.036e+15	—	—	—	3.036e+15	3.036e+15
100	0.50	**4.113e+15**	4.174e+15	4.176e+15	4.174e+15	4.174e+15	—	—	4.174e+15	—
200	1.00	5.542e+15	**5.407e+15**	5.409e+15	**5.407e+15**	—	—	—	**5.407e+15**	—
1000	5.00	4.72e+16	**2.282e+16**	—	—	—	—	a	a	—

— No solution found within the time limit

Table 2: Objective function for the same set of transportation instances with integer costs reported in Table 1; lowest in boldface.

n.var	n.cons	BlockIP Newton				BlockIP Pred-Corr				CPLEX HSD			CPLEX Barrier		
(M)	(M)	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	CPU	Obj.	It.
0.25	0.01	1.4	**6.284e+11**	65	148	**1.2**	6.285e+11	31	328	1.4	6.288e+11	24	1.4	**6.284e+11**	24
2.5	0.10	31.5	**6.27e+13**	147	489	**4.0**	6.536e+13	10	73	24.0	6.28e+13	38	13.6	6.271e+13	24
12.5	0.50	146.6	**1.567e+15**	141	343	**71.4**	**1.567e+15**	29	474	137.4	1.57e+15	42	79.1	**1.567e+15**	27
25	1.00	253.8	**6.268e+15**	122	299	**37.4**	6.687e+15	9	44	260.3	6.278e+15	39	153.7	6.269e+15	26
125	5.00	998.4	**3.499e+16**	85	391	**987.9**	**3.499e+16**	52	486	2267.6	3.501e+16	68	1019.0	3.512e+16	34
0.5	0.01	**2.0**	**4.401e+12**	48	174	2.9	4.402e+12	32	671	3.4	4.402e+12	30	2.4	4.402e+12	24
5	0.10	90.9	**4.382e+14**	151	749	37.1	**4.382e+14**	37	833	76.9	4.388e+14	64	**34.2**	**4.382e+14**	32
25	0.50	518.3	**9.91e+15**	200	1652	237.7	1.095e+16	50	942	478.4	1.097e+16	73	**221.0**	1.095e+16	37
50	1.00	966.3	**2.438e+16**	200	1314	519.0	2.56e+16	53	1049	1295.2	2.564e+16	94	**435.9**	2.56e+16	36
250	5.00	1578.2	**1.828e+16**	74	295	**1497.3**	**1.828e+16**	46	257	6245.7	1.829e+16	89	1980.5	**1.828e+16**	29
1	0.01	8.4	**3.643e+13**	63	1125	7.4	**3.643e+13**	41	887	6.6	3.644e+13	24	**6.5**	**3.643e+13**	29
10	0.10	131.3	**3.61e+15**	132	992	86.1	3.611e+15	39	1102	157.2	3.617e+15	62	**82.9**	3.611e+15	37
50	0.50	930.5	**3.77e+15**	200	1247	**519.6**	4.898e+15	58	1090	2424.4	4.899e+15	175	536.5	4.9e+15	41
100	1.00	2631.4	**6.854e+15**	200	1714	**771.9**	6.869e+15	48	671	4285.4	6.871e+15	150	851.2	6.869e+15	31
500	5.00	**3924.5**	**1.55e+16**	103	203	5979.6	**1.55e+16**	81	807	—	—	—	—		
2	0.01	31.2	**2.9e+14**	82	2696	28.2	**2.9e+14**	49	2636	26.4	**2.9e+14**	44	**16.0**	**2.9e+14**	29
20	0.10	389.4	**3.028e+15**	200	1323	693.2	3.036e+15	98	5929	1402.0	3.038e+15	247	**268.1**	3.036e+15	51
100	0.50	2194.7	**4.113e+15**	194	1951	**1116.1**	4.174e+15	65	1099	7209.0	4.176e+15	227	1138.7	4.174e+15	36
200	1.00	3629.8	5.542e+15	200	1172	**1411.6**	**5.407e+15**	49	409	15573.3	5.409e+15	237	2300.9	**5.407e+15**	34
1000	5.00	**2340.4**	4.72e+16	28	53	2706.9	**2.282e+16**	21	90	—	—	—	—		

— No solution found within the time limit

Table 3: Comparison of interior-point algorithms for the same transportation instances with integer costs reported in Table 1; best (lowest) CPU and objective in boldface.

n.var	n.cons	BlockIP		CPLEX			LEMON
(M)	(M)	Newton	PreCorr	HSD	Barrier	Dual S.	NetS
0.25	0.01	**0.9**	1.3	1.1	5.2	5.0	3.0
2.5	0.10	**8.8**	15.0	17.5	18.0	267.4	—
12.5	0.50	**46.5**	60.8	119.1	232.6	6742.8	—
25	1.00	**91.3**	118.4	228.6	695.4	18155.1	—
125	5.00	**485.7**	726.0	1609.3	3557.8	18027.7	—
0.5	0.01	3.8	**2.2**	2.3	6.7	10.2	2.4
5	0.10	44.6	41.7	**33.2**	48.8	1823.3	650.3
25	0.50	**101.4**	226.3	388.6	712.3	18431.2	—
50	1.00	**388.8**	489.6	913.8	675.2	18010.3	—
250	5.00	**1577.9**	1697.8	5500.0	5306.2	18029.4	—
1	0.01	**6.1**	8.1	6.8	16.3	29.4	—
10	0.10	**69.9**	87.5	152.0	221.3	5751.2	843.7
50	0.50	**349.7**	394.0	814.6	2665.9	18843.8	—
100	1.00	**804.9**	807.5	1947.7	2885.7	18189.8	—
500	5.00	**3993.9**	4006.9	—	—	24610.2	—
2	0.01	**17.7**	31.0	19.1	30.5	232.0	—
20	0.10	**162.3**	298.1	588.8	417.9	18029.0	839.2
100	0.50	2103.6	**1497.4**	3354.7	5825.6	18027.4	—
200	1.00	3979.8	**3072.5**	7023.6	11197.7	20801.6	—
1000	5.00	19401.8	**18803.4**	—	—	—	—

— No solution found within the time limit

Table 4: Computational performance for transportation instances with fractional costs; fastest execution in boldface. The first two columns describe the problem size in terms of the number of variables ($nm+n$) and constraints ($m+n$), while the remaining columns report the CPU time of MCCFBN-BlockIP, CPLEX and LEMON solvers.

n.var	n.cons	BlockIP		CPLEX			LEMON
(M)	(M)	Newton	PreCorr	HSD	Barrier	Dual S.	NetS
0.25	0.01	**6253**	**6253**	6254	**6253**	**6253**	**6253**
2.5	0.10	**6267**	6268	6275	6281	**6267**	—
12.5	0.50	**6268**	6269	6277	6288	**6268**	—
25	1.00	**6268**	6269	6279	6331	—	—
125	5.00	**6269**	**6269**	6278	6659	—	—
0.5	0.01	**4.351e+04**	4.358e+04	4.359e+04	4.358e+04	4.358e+04	4.358e+04
5	0.10	**4.371e+04**	**4.371e+04**	4.379e+04	4.393e+04	4.378e+04	4.378e+04
25	0.50	**4.325e+04**	4.374e+04	4.385e+04	4.564e+04	—	—
50	1.00	**4.373e+04**	**4.373e+04**	4.385e+04	4.395e+04	—	—
250	5.00	**4.373e+04**	4.375e+04	4.386e+04	4.51e+04	—	—
1	0.01	3.571e+05	**3.541e+05**	3.572e+05	3.593e+05	3.571e+05	—
10	0.10	3.603e+05	**3.585e+05**	3.608e+05	3.618e+05	3.603e+05	3.603e+05
50	0.50	**3.606e+05**	**3.606e+05**	3.612e+05	3.693e+05	—	—
100	1.00	**3.607e+05**	**3.607e+05**	3.613e+05	3.675e+05	—	—
500	5.00	**3.607e+05**	**3.607e+05**	—	—	—	—
2	0.01	**2.764e+06**	2.787e+06	2.788e+06	2.792e+06	2.787e+06	—
20	0.10	**2.795e+06**	2.824e+06	2.842e+06	2.844e+06	—	2.837e+06
100	0.50	**2.818e+06**	2.842e+06	2.847e+06	3.05e+06	—	—
200	1.00	**2.819e+06**	2.842e+06	2.847e+06	2.895e+06	—	—
1000	5.00	**2.819e+06**	2.841e+06	—	—	—	—

— No solution found within the time limit

Table 5: Objective function for the same transportation instances with fractional costs reported in Table 4; lowest in boldface.

Newton direction provided slightly faster executions, except for the largest instances, where predictor-corrector was superior. Table 5 supports the fact that all solvers provide points of similar objective function. Finally, Table 6 provides a closer look at the comparison between interior-point algorithms for the instances of Table 4, focusing on both CPU and objective function. It can be observed that, in average, the number of PCG iterations by IPM iteration was always below 20 for Newton and 50-60

n.var	n.cons	BlockIP Newton				BlockIP Pred-Corr				CPLEX HSD			CPLEX Barrier		
(M)	(M)	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	CPU	Obj.	It.
0.25	0.01	**0.9**	**6253**	37	196	1.3	**6253**	30	461	1.1	6254	15	5.2	**6253**	140
2.5	0.10	**8.8**	**6267**	36	196	15.0	6268	33	518	17.5	6275	28	18.0	6281	34
12.5	0.50	**46.5**	**6268**	38	198	60.8	6269	26	417	119.1	6277	36	232.6	6288	90
25	1.00	**91.3**	**6268**	38	170	118.4	6269	26	366	228.6	6279	34	695.4	6331	128
125	5.00	**485.7**	**6269**	38	178	726.0	**6269**	28	471	1609.3	6278	41	3557.8	6659	129
0.5	0.01	3.8	**4.351e+04**	77	706	**2.2**	4.358e+04	27	460	2.3	4.359e+04	19	6.7	4.358e+04	75
5	0.10	44.6	**4.371e+04**	84	710	41.7	**4.371e+04**	34	915	**33.2**	4.379e+04	24	48.8	4.393e+04	47
25	0.50	**101.4**	**4.325e+04**	41	254	226.3	4.374e+04	38	1124	388.6	4.385e+04	56	712.3	4.564e+04	125
50	1.00	**388.8**	**4.373e+04**	74	534	489.6	**4.373e+04**	39	1280	913.8	4.385e+04	62	675.2	4.395e+04	59
250	5.00	**1577.9**	**4.373e+04**	63	397	1697.8	4.375e+04	32	669	5500.0	4.386e+04	72	5306.2	4.51e+04	93
1	0.01	**6.1**	3.571e+05	51	710	8.1	**3.541e+05**	37	1089	6.8	3.572e+05	24	16.3	3.593e+05	80
10	0.10	**69.9**	3.603e+05	63	640	87.5	**3.585e+05**	41	1044	152.0	3.608e+05	58	221.3	3.618e+05	106
50	0.50	**349.7**	**3.606e+05**	66	610	394.0	**3.606e+05**	36	997	814.6	3.612e+05	56	2665.9	3.693e+05	225
100	1.00	**804.9**	**3.607e+05**	70	778	807.5	**3.607e+05**	37	863	1947.7	3.613e+05	63	2885.7	3.675e+05	119
500	5.00	**3993.9**	**3.607e+05**	69	739	4006.9	**3.607e+05**	38	923	—	—	—	—		
2	0.01	**17.7**	**2.764e+06**	87	838	31.0	2.787e+06	40	3293	19.1	2.788e+06	31	30.5	2.792e+06	60
20	0.10	**162.3**	**2.795e+06**	78	666	298.1	2.824e+06	50	2416	588.8	2.842e+06	100	417.9	2.844e+06	79
100	0.50	2103.6	**2.818e+06**	133	2673	**1497.4**	2.842e+06	46	1998	3354.7	2.847e+06	101	5825.6	3.05e+06	207
200	1.00	3979.8	**2.819e+06**	135	2480	**3072.5**	2.842e+06	47	2245	7023.6	2.847e+06	102	11197.7	2.895e+06	192
1000	5.00	19401.8	**2.819e+06**	136	2269	**18803.4**	2.841e+06	54	2744	—	—	—	—		

— No solution found within the time limit

Table 6: Comparison of barrier solvers for the same transportation instances with fractional costs reported in Table 4; best (lowest) CPU and objective in boldface.

for predictor-corrector. As for the integer cost instances, CPLEX could not solve the largest instances within the time limit.

5.4 Transportation instances with quadratic and fractional costs

To enlarge the level of generalization of the analyzed MCCFBN instances, quadratic costs can be included to model congestion (in urban transportation settings), line resistance (in power grids and electricity distribution) or inefficiency of scale (in labor-machine assignments). Thus, we consider again the battery of computational tests presented Subsection 5.2, with the only difference given by the inclusion of a quadratic convex term in the objective functions. As a results of this modeling design, only interior-point algorithms (MCCFBN-BlockIP and CPLEX) are applicable. Table 7 provides the instances dimensions, CPU time, objective function and IPM iterations of MCCFBN-BlockIP and CPLEX quadratic solvers; for MCCFBN-BlockIP the overall number of PCG iterations is also reported.

For this class of instances MCCFBN-BlockIP outperformed the two interior-point variants of CPLEX in all the cases. Table 7 shows again a larger number of IPM iterations by MCCFBN-BlockIP, while remaining globally more efficient than the CPLEX barrier across the entire collection of MCCFBN problem instances. We also observe that the number of IPM and PCG iterations needed by MCCFBN-BlockIP is much less than for linear instances; this is consistent with Proposition 2 and the computational results of [10].

5.5 Assignment instances

As discussed in Section 1, assignment instances are a particular class of MCCFBN problems, which compute a minimum cost matching between two groups of objects, where nodes in one layer are associated to demands for a commodity or task to be performed (negative flow injection), whereas nodes in the other layer represent the corresponding suppliers or agents (positive flow injection). Thus, in the case of assignment problems variables are bounded between zero and one, while the right-hand-term is composed by unitary elements. Due to the total unimodularity structure, the optimal solution is binary with simplex algorithms. Using an IPM this is not guaranteed, unless a crossover postprocess is applied; however, we observed that only a small fraction of flows resulted fractional in the optimal point. In the computational results of this section no crossover was considered. All MCCFBN-BlockIP, CPLEX and LEMON solvers were used to solve the set of assignment instances obtained with the previously mentioned DIMACS generator [18]. We generated two groups of assignment instances, with two different maximum costs, 5000 and 5000000, in an attempt to avoid favoring the cost scaling algorithm of LEMON (which is pseudo-polynomial in the maximum cost). And for each group we generated six instances considering $n,m \in \{250,500,2500,5000,12500,25000,\}$, which amounts to assignment problems of up to 625M variables and 50000 constraints.

n.var	n.cons	BlockIP Newton				BlockIP Pred-Corr				CPLEX HSD			CPLEX Barrier		
(M)	(M)	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	CPU	Obj.	It.
0.25	0.01	**0.4**	**171.7**	17	45	**0.4**	**171.7**	10	51	0.7	171.8	9	0.7	**171.7**	13
2.5	0.10	**4.2**	**17.21**	18	46	4.3	**17.21**	11	54	7.6	**17.21**	9	10.8	**17.21**	18
12.5	0.50	22.6	**3.443**	19	52	**20.7**	**3.443**	11	55	45.5	**3.443**	10	65.6	**3.443**	18
25	1.00	48.4	**1.722**	20	53	**42.5**	**1.722**	11	56	91.0	**1.722**	10	140.1	**1.722**	20
125	5.00	230.2	**0.3444**	20	54	**202.8**	**0.3444**	11	56	489.9	**0.3444**	10	770.1	**0.3444**	22
0.5	0.01	**0.8**	**4757**	19	50	0.9	**4757**	14	56	1.5	**4757**	10	1.4	**4757**	12
5	0.10	**9.9**	**477.8**	23	52	**9.9**	**477.8**	15	55	20.3	**477.8**	13	21.7	**477.8**	17
25	0.50	**51.3**	**95.6**	24	53	52.0	**95.6**	15	61	104.9	**95.6**	11	148.3	**95.6**	20
50	1.00	**114.3**	**47.8**	27	56	119.7	**47.8**	18	71	214.0	**47.8**	10	302.9	**47.8**	21
250	5.00	583.8	**9.561**	27	64	**505.6**	**9.561**	15	62	1119.4	**9.561**	11	1685.6	**9.561**	22
1	0.01	**2.4**	**1.588e+05**	29	48	3.1	**1.588e+05**	25	106	4.0	**1.588e+05**	14	3.8	**1.588e+05**	14
10	0.10	**24.9**	**1.603e+04**	24	50	29.1	**1.603e+04**	22	92	48.2	**1.603e+04**	16	48.2	**1.603e+04**	18
50	0.50	**113.2**	**3208**	27	57	157.9	**3208**	24	102	300.6	**3208**	17	325.4	**3208**	21
100	1.00	**222.0**	1604	26	63	262.8	**1603**	20	79	483.0	1604	12	713.5	1604	23
500	5.00	**1254.3**	**320.8**	28	62	1415.2	**320.8**	21	83	—	—	—	—		
2	0.01	**5.8**	**4.88e+06**	36	69	7.1	**4.88e+06**	29	106	11.2	**4.88e+06**	17	9.3	**4.88e+06**	14
20	0.10	**74.7**	4.971e+05	46	63	115.6	**4.968e+05**	34	150	111.2	**4.968e+05**	15	112.6	**4.968e+05**	17
100	0.50	**334.5**	9.952e+04	40	76	450.3	**9.951e+04**	34	156	749.3	9.952e+04	19	816.4	9.952e+04	23
200	1.00	**615.3**	**4.977e+04**	37	67	743.6	**4.977e+04**	28	128	1514.1	**4.977e+04**	18	1935.9	**4.977e+04**	27
1000	5.00	**3498.6**	**9956**	39	72	4819.9	**9956**	31	133	—	—	—	—		

— No solution found within the time limit

Table 7: Comarison of barrier solvers for transportation instances with quadratic costs. The first two columns describe the problem size in terms of the number of variables ($nm+n$) and constraints ($m+n$), while the remaining columns report the CPU time, objective function, and IPM iterations of MCCFBN-BlockIP and CPLEX quadratic solvers. For MCCFBN-BlockIP the number of PCG iterations is also provided. Best (lowest) CPU and objective in boldface.

n.var	n.cons	BlockIP		CPLEX			LEMON			
(M)		Newton	PreCorr	HSD	Barrier	Dual S.	CapS	CosS	CycC	NetS
0.0625	500	0.1	0.1	0.3	0.2	0.1	**0.0**	**0.0**	0.3	**0.0**
0.25	1000	0.5	0.4	1.2	1.1	0.6	0.1	0.1	1.2	**0.0**
6.25	5000	34.3	38.1	92.4	78.4	90.3	**1.6**	5.6	77.4	1.7
25	10000	216.1	194.1	741.9	686.5	331.1	**8.5**	23.8	435.0	8.7
156.25	25000	1507.8	1222.6	7853.5	8586.4	2079.9	129.4	167.4	3827.5	**70.8**
625	50000	4031.8	4165.5	—	—	18182.1	4191.2	436.2	11104.2	**240.7**
0.0625	500	**8209**	**8209**	**8209**	**8209**	**8209**	**8209**	**8209**	**8209**	**8209**
0.25	1000	**8323**	**8323**	**8323**	8324	**8323**	**8323**	**8323**	**8323**	**8323**
6.25	5000	**9415**	**9415**	**9415**	**9415**	**9415**	**9415**	**9415**	**9415**	**9415**
25	10000	**1.09e+04**	**1.09e+04**	**1.09e+04**	**1.09e+04**	**1.09e+04**	**1.09e+04**	**1.09e+04**	**1.09e+04**	**1.09e+04**
156.25	25000	**1.521e+04**	**1.521e+04**	**1.521e+04**	**1.521e+04**	**1.521e+04**	**1.521e+04**	**1.521e+04**	**1.521e+04**	**1.521e+04**
625	50000	**2.54e+04**	**2.54e+04**	—	—	—	**2.54e+04**	**2.54e+04**	**2.54e+04**	**2.54e+04**

— No solution found within the time limit

Table 8: CPU time (six first rows) and objective function (last six rows) for assignment instances with maximum cost of 5000; fastest execution and lowest objective in boldface.

n.var	n.conss	BlockIP		CPLEX			LEMON			
(M)		Newton	PreCorr	HSD	Barrier	Dual S.	CapS	CosS	CycC	NetS
0.0625	500	0.2	0.1	0.3	0.2	0.1	**0.0**	**0.0**	0.2	**0.0**
0.25	1000	0.6	0.4	1.4	1.3	0.6	0.1	0.2	1.0	**0.0**
6.25	5000	33.3	24.9	98.1	90.1	27.0	2.0	4.9	83.1	**1.9**
25	10000	146.5	130.1	739.8	832.6	166.0	11.9	25.7	540.2	**9.5**
156.25	25000	1499.3	1017.0	8805.4	12959.6	1380.7	111.3	327.5	5339.7	**83.4**
625	50000	8047.6	7525.8	—	—	18141.8	609.2	935.4	—	**492.7**
0.0625	500	**8.088e+06**	**8.088e+06**	**8.088e+06**	**8.088e+06**	**8.088e+06**	**8.088e+06**	**8.088e+06**	**8.088e+06**	**8.088e+06**
0.25	1000	**8.089e+06**	**8.089e+06**	8.09e+06	**8.089e+06**	**8.089e+06**	**8.089e+06**	**8.089e+06**	**8.089e+06**	**8.089e+06**
6.25	5000	**8.133e+06**	**8.133e+06**	**8.133e+06**	**8.133e+06**	**8.133e+06**	**8.133e+06**	**8.133e+06**	**8.133e+06**	**8.133e+06**
25	10000	**8.28e+06**	**8.28e+06**	**8.28e+06**	**8.28e+06**	**8.28e+06**	**8.28e+06**	**8.28e+06**	**8.28e+06**	**8.28e+06**
156.25	25000	**8.208e+06**	**8.208e+06**	**8.208e+06**	**8.208e+06**	**8.208e+06**	**8.208e+06**	**8.208e+06**	**8.208e+06**	**8.208e+06**
625	50000	8.206e+06	8.206e+06	—	—	—	**8.205e+06**	**8.205e+06**	—	**8.205e+06**

— No solution found within the time limit

Table 9: CPU time (six first rows) and objective functions (last six rows) for assignment instances with maximum cost of 5000000; fastest execution and lowest objective in boldface.

Tables 8 and 9 compare MCCFBN-BlockIP, CPLEX and LEMON solvers with respect to CPU time (first six rows of each table) and objective function (last six rows), for the two groups of assignment instances, respectively.

In both cases, and unlike for transportation problems, the specialized LEMON algorithms outperformed both CPLEX and MCCFBN-BlockIP. And the network simplex implementation of LEMON (instead of the cycle canceling, capacity or cost scaling pseudo-polynomial algorithms) was the most efficient option for the largest problems; this is consistent with the results provided in [20]. All solvers achieved the same optimal values. Finally, Table 10 reports a comparison between the IPM solvers. Clearly, MCCFBN-BlockIP was more efficient than CPLEX for all the instances. If we had solved assignment instances with quadratic terms (should they were useful to model some particular problem), the only available algorithms would be those in MCCFBN-BlockIP and CPLEX; and since MCCFBN-BlockIP is more efficient for quadratic than for linear problems, we would expect it to outperform CPLEX even by a larger margin.

n.var	n.cons	BlockIP Newton				BlockIP Pred-Corr				CPLEX HSD			CPLEX Barrier		
(M)		CPU	Obj.	It.	PCG It	CPU	Obj.	It.	PCG It	CPU	Obj.	It.	CPU	Obj.	It.
0.0625	500	**0.1**	**8209**	17	121	**0.1**	**8209**	11	156	0.3	**8209**	12	0.2	**8209**	13
0.25	1000	0.5	**8323**	20	148	**0.4**	**8323**	11	153	1.2	**8323**	13	1.1	8324	14
6.25	5000	**34.3**	**9415**	32	855	38.1	**9415**	20	1031	92.4	**9415**	14	78.4	**9415**	13
25	10000	216.1	**1.09e+04**	40	1503	**194.1**	**1.09e+04**	22	1385	741.9	**1.09e+04**	17	686.5	**1.09e+04**	17
156.25	25000	1507.8	**1.521e+04**	49	1493	**1222.6**	**1.521e+04**	23	1295	7853.5	**1.521e+04**	14	8586.4	**1.521e+04**	16
625	50000	**4031.8**	**2.54e+04**	63	318	4165.5	**2.54e+04**	30	453	—	—	—	—		
0.0625	500	0.2	**8.088e+06**	19	139	**0.1**	**8.088e+06**	12	159	0.3	**8.088e+06**	11	0.2	**8.088e+06**	13
0.25	1000	0.6	**8.089e+06**	22	156	**0.4**	**8.089e+06**	14	145	1.4	8.09e+06	13	1.3	**8.089e+06**	15
6.25	5000	33.3	**8.133e+06**	33	475	**24.9**	**8.133e+06**	20	479	98.1	**8.133e+06**	15	90.1	**8.133e+06**	15
25	10000	146.5	**8.28e+06**	40	773	**130.1**	**8.28e+06**	22	721	739.8	**8.28e+06**	18	832.6	**8.28e+06**	22
156.25	25000	1499.3	**8.208e+06**	51	1059	**1017.0**	**8.208e+06**	25	868	8805.4	**8.208e+06**	16	12959.6	**8.208e+06**	25
625	50000	8047.6	**8.206e+06**	55	1769	**7525.8**	**8.206e+06**	27	1811	—	—	—	—		

— No solution found within the time limit

Table 10: Comparison of barrier solvers. The upper part of the table contains assignment instances with maximum cost of 5000, whereas the bottom part contains assignment instances with maximum cost of 5000000. Best (lowest) CPU and objective are reported in boldface.

6 Conclusions

In this work we exploited the scalable properties of a specialized interior-point method for solving huge instances of minimum convex cost flow problems in bipartite networks. This was accomplished by taking advantage of a particularly suitable primal block-angular structure of the underlying constraints matrix, which allows for an extremely efficient computation of the Newton direction at each IPM iteration.

This scalable properties have been theoretically studied, building on the asymptotic behavior of the inner PCG procedure for MCCFBN problems. This allows the minimum convex cost flows problems in bipartite networks to be included within the successful applications of the specialized IPM for primal block-angular problems.

The tests performed and reported in the paper involve an extensive set of computational experiments with linear and quadratic instances of up to 1 billion arcs and 200 and 5 million nodes in each subset of the node partition. The results support the competitive advantage of the specialized IPM for MCCFBN problems, which is particularly true for the case of quadratic objective costs.

Overall, the specialized IPM significantly outperformed the other approaches in most of the linear and quadratic transportation instances tested. In particular, it always provided a solution within the five hours time limit and it never exhausted the 192 Gigabytes of memory of the server used for the runs. Given the massive amount of data being collected in the current digital society, in the near future we may need to solve extremely huge network optimization problems (even larger than those tried in this work). This specialized IPM for MCCFBN may be considered a good candidate for those problems, if not one of the few methods able to tackle them.

Acknowledgements The first author has been supported by the MINECO/FEDER grant MTM2015-65362-R.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. Prentice-Hall, Englewood Cliffs, NJ (1993)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B., Reddy, M.R.: Handbooks in Operations Research and Management Science, vol. 7, chap. Applications of network optimization, pp. 1–83. Elsevier (1995)
3. Bellavia, S., Gondzio, J., Morini, B.: A matrix-free preconditioner for sparse symmetric positive definite systems and least-squares problems. SIAM Journal on Scientific Computing **35**(1), A192–A211 (2013)
4. Bergamaschi, L., Gondzio, J., Zilli, G.: Preconditioning indefinite systems in interior point methods for optimization. Computational Optimization and Applications **28**(2), 149–171 (2004)
5. Boland, N., Christiansen, J., Dandurand, B., Eberhard, A., Oliveira, F.: A parallelizable augmented Lagrangian method applied to large-scale non-convex-constrained optimization problems. Mathematical Programming (2018)

6. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems, Revised Reprint. SIAM, Philadelphia, PA (2012)
7. Cao, Y., Laird, C.D., Zavala, V.M.: Clustering-based preconditioning for stochastic programs. Computational optimization and applications **64**(2), 379–406 (2016)
8. Castro, J.: A specialized interior-point algorithm for multicommodity network flows. SIAM Journal on Optimization **10**, 852–877 (2000)
9. Castro, J.: Interior-point solver for convex separable block-angular problems. Optimization Methods and Software **31**, 88–109 (2016)
10. Castro, J., Cuesta, J.: Quadratic regularizations in an interior-point method for primal block-angular problems. Mathematical Programming **130**, 415–445 (2011)
11. Castro, J., Nasini, S.: On geometrical properties of preconditioners in IPMs for classes of block-angular problems. SIAM Journal on Optimization **27**(3), 1666–1693 (2017)
12. Castro, J., Nasini, S., Saldanha-da Gama, F.: A cutting-plane approach for large-scale capacitated multi-period facility location using a specialized interior-point method. Mathematical Programming **163**, 411–444 (2017)
13. Curtis, F.E., Jiang, H., Robinson, D.P.: An adaptive augmented Lagrangian method for large-scale constrained optimization. Mathematical Programming **152**(1), 201–245 (2015)
14. Frangioni, A., Gentile, C.: New preconditioners for KKT systems of network flow problems. SIAM Journal on Optimization **14**(3), 894–913 (2004)
15. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by canceling negative cycles. Journal of the ACM **36**(4), 873–886 (1989)
16. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by successive approximation. Mathematics of Operations Research **15**(3), 430–466 (1990)
17. Gondzio, J.: Convergence analysis of an inexact feasible interior point method for convex quadratic programming. SIAM Journal on Optimization **23**(3), 1510–1527 (2013)
18. Johnson, D.S., McGeoch, C.C. (eds.): Network Flows and Matching: First DIMACS Implementation Challenge. American Mathematical Society, Providence, RI (1993)
19. Klein, M.: A primal method for minimal cost flows with applications to the assignment and transportation problems. Management Science **14**(3), 205–220 (1967)
20. Kovács, P.: Minimum-cost flow algorithms: an experimental evaluation. Optimization Methods and Software **30**(1), 94–127 (2015)
21. Lee, H., Garcia-Diaz, A.: A network flow approach to solve clustering problems in group technology. The International Journal of Production Research **31**(3), 603–612 (1993)
22. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. Combinatorica **7**(1), 105–113 (1987)
23. Mulvey, J.M.: Testing of a large-scale network optimization program. Mathematical Programming **15**(1), 291–314 (1978)

24. Nesterov, Y.: Introductory Lectures on Convex Optimization: A Basic Course, vol. 87. Kluwer (2004)
25. Oliveira, A.R., Sorensen, D.C.: A new class of preconditioners for large-scale linear systems from interior point methods for linear programming. Linear Algebra and its applications **394**, 1–24 (2005)
26. Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Mathematical Programming **78**(2), 109–129 (1997)
27. Resende, M.G., Veiga, G.: An implementation of the dual affine scaling algorithm for minimum-cost flow on bipartite uncapacitated networks. SIAM Journal on Optimization **3**(3), 516–537 (1993)
28. Wright, S.: Primal-Dual Interior-Point Methods. SIAM, Philadelphia, PA (1996)
29. Zangwill, W.I.: Minimum concave cost flows in certain networks. Management Science **14**(7), 429–450 (1968)