**ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA**
**CONSIGLIO NAZIONALE DELLE RICERCHE**

**A. Frangioni, C. Gentile**

# INTERIOR POINT METHODS FOR NETWORK PROBLEMS

**R. 539 Dicembre 2000**

**Antonio Frangioni** – Dipartimento di Informatica, Corso Italia 40, 56125 Pisa (Italy). Email: `frangio@di.unipi.it`.

**Claudio Gentile** – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30 - 00185 Roma, Italy. Email: `gentile@iasi.rm.cnr.it`.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica, CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: `iasi@iasi.rm.cnr.it`
URL: `http://www.iasi.rm.cnr.it`

## Abstract

We propose a new set of preconditioners for the iterative (approximate) solution, via a Preconditioned Conjugate Gradient (PCG) method, of the "critical" linear systems that has to be solved at each iteration of an Interior Point (IP) algorithm for the solution of Linear Min Cost Flow (MCF) problems. These preconditioners are based on the idea of extracting a proper subgraph of the original graph which contains (possibly strictly) a spanning tree. Computational results will be presented in a companion paper.

*Key words:* Min Cost Flow problems, Interior Point algorithms, Preconditioned Conjugated Gradient method.

## 1. Introduction

The Linear Min Cost Flow (MCF) problem is the following $LP$

$$min\{ \ cx \ : \ Ex = b \ , \ 0 \leq x \leq u \ \} \ , \tag{1}$$

where $E$ is the node-arc incidence matrix of a network $G = (N, A)$, $c$ is the vector of arc costs, $u$ is the vector of arc upper capacities, $b$ is the vector of node deficits, and $x$ is the vector of flows. This problem has a huge set of applications, either in itself or – more often – as a submodel of more complex and demanding problems [1]. This is testimonied by the enormous amount of research that have been invested in defing efficient solution algorithms for MCF problems [1], either by specializing LP algorithms – such as the Simplex method – to the network case, or by developing ad-hoc approaches.

Recently, Interior Point (IP) methods for Linear Programming [13, 16] have grown a well-established reputation as efficient algorithms for large-scale problems. In these methods, at each iteration we have to solve linear systems of the form

$$(E\Theta E^T)\Delta y = d \tag{2}$$

where $\Theta$ and $d$ are respectively a $m \times m$ diagonal matrix ($m = |A|$) with positive entries and a vector of $\mathbb{R}^n$ ($n = |N|$) which depend on the current solution and on the IP algorithm chosen. In most general-purpose LP solvers, these linear systems are solved by means of direct methods, typically the Cholesky factorization preceded by a heuristic reordering of the columns of $E$ aimed at minimizing the "fill-in" [13]. For structured LPs like MCF problems, however, this approach fails to be efficient enough because the computational burden of the Cholesky factorization is too high. Alternative methods have been proposed [3, 12, 10], not only for MCFs, where the system is solved using a Preconditioned Conjugate Gradient (PCG) method. In all these methods, the critical choice is that of the preconditioner: on one side, it must be inexpensive to compute and invert, while on the other side it has to deliver a consistent reduction of the number of CG iterations required to (approximately) solve the system (2). Some proposals for preconditioners, mostly based on (approximate) Minimum Spanning Tree computations, can be found in the literature [12, 10]; the resulting IP algorithms are shown to be computationally on par, on some classes of instances, with the most efficient combinatorial algorithms.

Our aim is to improve the effectiveness of IP methods for MCF problems by designing new classes of efficient preconditioners. The basic idea is that of extracting a proper subgraph $S = (N, A_S)$ of $G$ ($A_S \subseteq A$) which contains – possibly strictly – a spanning tree, but such that the resulting matrix

$$M_S = E_S \Theta_S E_S^T \ , \tag{3}$$

where $E_S$ and $\Theta_S$ denote respectively the matrices $E$ and $\Theta$ restricted to the columns relative to the arcs in $S$, can be efficiently factorized. Other ideas can then be exploited for further improving the effectiveness of these preconditioners without increasing the computational cost of the algorithms involved.

Since a large variety of preconditioners can be obtained in this way, computational experiences are necessary to tell which combination provides the better trade-off between the cost of finding $S$ and factorizing $M_S$, on one side, and the overall number of PCG iterations on the other. Finding the "best" preconditioner is further complicated by the fact that a number of different IP algorithms exist (primal, dual, primal-dual ...), possibly with several variants each (affine, barrier, predictor-corrector ...). Since the impact of the approximate solution of the systems (2) on the different algorithmic variants can be different, leading to a different number of overall IP iterations, computational experiences are required in order to find the mix which delivers the best overall performances. Finally, alternative preconditioners can have very different performances depending on the structure of the instance, such as the topology of the underlying graph. Therefore, an extensive set of computational results is required in order to properly assess the effectiveness of the proposed preconditioners; for the sake of clarity, those computational results will be presented in a companion paper.

The structure of the paper is the following: in section 2 (several variants of) Interior Point algorithms are introduced, and the relevant algorithmic issues are discussed. In section 3 the preconditioners for MCF problems proposed in the literature are reviewed and studied, while in section 4 we introduce and prove the properties of a large family of new preconditioners. Then, in Section 5 and 6 the computational issues related to this new families of preconditioners are discussed. Finally, in section 7 conclusions are drawn.

## 2. Interior Point algorithms

Interior Point algorithms for MCF can be described by considering (1) rewritten as

$$\min\{\, cx \ :\ Ex = b,\ x + s = u,\ x, s \geq 0 \,\}\ , \tag{4}$$

where $x \in \mathbb{R}^m$ and $s \in \mathbb{R}^m$ are respectively the primal variables and the slacks of the box constraints, and its dual

$$\max\{\, yb - wu \ :\ yE + z - w = c,\ z, w \geq 0 \,\}\ , \tag{5}$$

where $y \in \mathbb{R}^n$, $z \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$ are respectively the dual variables of the structural constraints $Ex = b$, the dual slacks and the dual variables of the box constraints $x \leq u$.

A number of different IP algorithms can be constructed, which all start from the "slackened" version of the *KKT optimality conditions* of this dual pair of problems

$$Ex = b \tag{6}$$

$$yE + z - w = c \tag{7}$$

$$x + s = u \tag{8}$$

$$XZe = \mu e \tag{9}$$

$$SWe = \mu e \tag{10}$$

$$(x, s, z, w) \geq 0$$

where $\mu$ is a parameter, $e$ is the vector of 1's of proper dimension, and each uppercase letter corresponds to the diagonal matrix having as diagonal elements the entries of the corresponding lowercase vector. For $\mu = 0$, the above system characterizes all the optimal solutions of (4) and (5). For $\mu > 0$, the unique solution of the system (6) – (10) lies on the *central path*, a continuous trajectory which, as $\mu$ tends to 0, converges to an optimal solution of (4) and (5) (more precisely, it converges to the analytic centers of the primal and dual optimal faces).

*Path-following*, also known as *barrier*, algorithms attempt to reach close to these optimal solutions by following the central path. This is done by performing a damped version of Newton's iteration applied to the nonlinear system (6) – (10). These algorithms can be divided into *primal*, *dual* or *primal-dual* according to how exactly the Newton step is done.

In the primal method, a primal (not necessarily feasible) point $(x, s)$ is kept as the center of the Newton iteration, and dual variables $(y, z, w)$ corresponding to the direction $(\delta x, \delta s)$ are derived from proper linearizations of the equations of the system. More precisely, from (9) one obtains

$$z = \mu[X + \delta X]^{-1}e,$$

which is then linearized with a first-order Taylor expansion as

$$z = \mu[X^{-1}e - X^{-2}\delta x].$$

A similar formula for $w$ can be obtained by linearizing (10), and by using these relations in the remaining equations one obtains explicit formulae for $y$, $\delta x$ and $\delta s$. These formulae can alternatively be seen (and are usually derived) as the formulae for the Newton step of a nonlinear version of the primal problem where the constraints $0 \leq x \leq u$ are replaced with the logarithmic barrier term $-\mu(\sum_i \ln(x_i) + \sum_i \ln(u_i - x_i))$.

In the dual method, the dual point $(y, z, w)$ is kept as the center of the Newton iteration, and primal variables $(x, s)$ corresponding to the direction $(\delta y, \delta z, \delta w)$ are derived from proper linearizations of the equations of the system. From the first-order Taylor expansion of (9) one obtains

$$x = \mu[Z^{-1}e - Z^{-2}\delta z],$$

for which explicit formulae for $s$ and $(\delta y, \delta z, \delta w)$ can be derived. These formulae can be seen as the formulae for the Newton step of a nonlinear version of the dual problem where the constraints $z \geq 0$ and $w \geq 0$ are replaced with the logarithmic barrier term $-\mu(\sum_i \ln(z_i) + \sum_i \ln(w_i))$.

Finally, in the primal-dual method both a primal point $(x, s)$ and a dual point $(y, z, w)$ are kept as the center of the Newton iteration, and a primal-dual Newton direction $(\delta x, \delta s, \delta y, \delta z, \delta w)$ is sought for. This is obtained by rewriting (9) – (10) as

$$(X + \delta X)(Z + \delta Z)e = \mu e$$

$$(S + \delta S)(W + \delta W)e = \mu e$$

and then linearizing the above nonlinear system by dropping the second-order terms $\delta X \delta Z$ and $\delta S \delta W$.

Once that a Newton direction – in the primal and/or dual space – has been obtained, an appropriate stepsize is selected which moves the current point "closer" to the central path for the current value of $\mu$, then $\mu$ is reduced by multiplying it for a factor $\rho < 0$ and the whole process is repeated. With appropriate choices of the stepsize, a sequence of primal and/or dual points converging to an optimal solution is constructed.

Several variants of the above methods have been defined. For instance, in the *predictor-corrector* variant of the primal-dual method an iterative approach is taken for refining the solution of the nonlinear system by iteratively substituting the obtained values of $\delta X$, $\delta Z$, $\delta S$ and $\delta W$ in the neglected quadratic terms and re-solving the modified linear system; this comes at the cost of multiple solutions of the system for each iteration, but usually improves on the number of overall IP iterations.

Also, *affine* variants of the above IP algorithms have been developed, where the formulae for the Newton direction are taken as the limit for $\mu \rightarrow 0$ of the formulae in the path-following case; this simplifies the formulae, making them faster to compute and completely eliminating $\mu$ from them, thereby avoiding the problem of tuning its decrease. On the other hand, affine variants tend to be less numerically stable than barrier variants.

A more detailed description of the IP algorithms and their underlying theory can be found in many linear programming textbooks, e.g. [13, 16].

Remarkably, all the formulae for all the variants of the IP method boil down to a set of linear operations plus one (or more) solution(s) of a "core" linear system of the form (2), where $\Theta$ and $d$ depend on which variant of IP algorithm is used. The solution of (2) typically represents by far the main computational burden of the IP algorithms. Thus, developing a specialized approach for the solution of (2) for specially-structured matrices $E$ can substantially improve the performances of an IP method [3, 12, 10]. Also, since the form of the "core" system is independent from the specific variant of IP algorithm used, the same specialized solver for (2) can be used to implement all the variants of IP algorithms.

Since $M = E\Theta E^T$ is symmetric and positive definite, the linear system (2) is usually solved through a Cholesky factorization, which is computationally effective and numerically stable. That is, a lower triangular *Cholesky factor* $L$ with all diagonal entries equal to 1 and a diagonal matrix $D > 0$ are found such that $M = LDL^T$; this can be done on $O(n^3)$, and, once that the factorization has been computed, systems involving $M$ can be solved in $O(n^2)$ with two backsolves on $L$. However, a well-known drawback of the Cholesky factorization is the *fill-in* phenomenon: a sparse matrix $M$ may have a dense Cholesky factor $L$. The density of the Cholesky factor may vary by reordering the rows and the columns of the matrix $E$; hence, IP codes usually make an effort for finding a permutation of the columns of $E$ which minimizes the fill-in effect. This needs only to be done at the beginning of the algorithm, since the structure of the nonzeroes in $M$ (and, therefore, of its Cholesky factor) does not depend on $\Theta$, and therefore does not change with the iterations. Unfortunately, the problem of finding the reordering which produces the least fill-in is known to be $NP$-hard [14]; however, several effective heuristics have been developed for computing a "good" such permutation [13].

Yet, in general the fill-in phenomenon cannot be avoided [3], except in some specific cases, so that for sparse matrices iterative methods such as the PCG may be more efficient than the Cholesky factorization. The following sections will be devoted to the study of the preconditioners for the solution of (2) that have been proposed in the literature, and to the proposal of new ones.

## 3. Preconditioners for MCF problems

The first PCG-based IP algorithm specifically taylored for MCF problems was proposed in [12]. Following suggestions from [6] and [15], the *tree-based preconditioner* was proposed, which is a preconditioner of the form (3) where $S$ is a spanning tree $T$ of $G$. In particular, $T$ is chosen as a(n approximate) maximum-weight spanning tree, the weight of each arc $(i, j)$ being the corresponding $\theta_{ij}$. Such a tree can be constructed in $O(m)$ with a variant of the classical Kruskal algorithm where arcs are only approximately sorted using a "bucket" data structures with $m$ buckets.

The tree-based preconditioner is computationally efficient. In fact, in the classical implementation of the PCG method, it is required to solve a linear system of the form $M_S z = r$; in the tree-based case, this can be done in $O(n)$ considering three linear systems: the first with coefficient matrix $E_S$, the second with $\Theta_S$ and the third with $E_S^T$. It is well-known [1] that systems with the matrices $E_S$ and $E_S^T$ can be solved in $O(n)$ by means of visits on the tree $T$.

The preconditioner can also be expected to be spectrally effective, especially in the last iterations of an interior-point algorithm. In fact, the analisys of interior-point methods shows that, if the optimal solution of the underlying problem is unique, then the weights $\theta_{ij}$ tend to zero on all arcs but those corresponding to the basic optimal solution [13], that form a spanning tree. Hence, in the last iterations of the IP method $M_S \approx E\Theta E^T$, and therefore the preconditioner can be expected to be very efficient. Let us remark that a slightly more formal rationale for the choice of a maximum-weight spanning tree for $S$ has been given in [5], where it is shown that

$$\kappa_2((E_S E_S^T)^+ E\Theta E^T) \geq \theta_{\min}(S)$$

where $X^+$ denotes the pseudo inverse of Moore-Penrose of a matrix $X$, $\kappa_2$ denotes the spectral condition number and $\theta_{\min}(S)$ denotes the minimum value $\theta_i$ among all arcs $i$ in $S$ (equivalently, the minimum diagonal element in $\Theta_S$). Interestingly, the Kruskal algorithm that is typically used for computing the maximum-weight $S$ also gives the tree with largest minimum weight over the arcs, and therefore it maximizes (the above estimate of) $\theta_{\min}(S)$.

Indeed, the experiments show that tree-based preconditioners are effective in the latest iterations of the IP method; unfortunately, they are less so in the first ones. This suggests an hybrid preconditioning technique [12], where the diagonal preconditioner is used in the first iterations, and then some heuristic rules are used to switch to the tree-based preconditioner in a later stage. The implementation of this approach, refined with better stop criteria [11] and a custom primal-feasible/ dual-infeasible IP algorithm [10], has shown to be competitive with well-known combinatorial MCF codes.

A different preconditioner has been proposed in [8] for the special case of transportation problems, and then extended to the general MCF problem in [9]. This preconditioner is based on the idea that the Cholesky factorization of $M = E\Theta E^T$ can be obtained by computing the QR factorization of the matrix

$$\bar{E}^T = \begin{bmatrix} \Theta_S^{1/2} & E_S^T \\ \Theta_{\bar{S}}^{1/2} & E_{\bar{S}}^T \end{bmatrix}$$

where $\bar{S} = A \setminus S$. In fact, given a QR factorization of $\bar{E}$, i.e., a triangular matrix $R$ and a unitary matrix $Q$ such that $\bar{E} = QR$, it is easy to check that

$$E\Theta E^T = E\Theta^{1/2}\Theta^{1/2}E^T = \bar{E}^T\bar{E} = R^T Q^T Q R = R^T R \ .$$

The QR factorization of $\bar{E}$ is not less expensive than the Cholesky factorization of $M$, and it usually causes fill-in destroying the nonzero pattern of $\bar{E}$. Thus, an *incomplete QR factorization* is used, where the Givens rotations are carefully selected to avoid the fill-in effect; this incomplete factorization is used as a preconditioner.

On transportation problems, this preconditioner has been reported [8] to be superior to the tree preconditioner, in particular by being effective even in the early iterations of the algorithm; however, these findings have been challenged in [9] for the case of general MCF problems.

Finally, another preconditioner has been proposed in [7]; the idea is to "extend" the tree-based preconditioner $M_T$ by using

$$M_T' = M_T + \rho \; diag(M - M_T) \tag{11}$$

as the preconditioner, where $diag(A)$ is the diagonal matrix having as the diagonal elements those of $A$. This has the advantage of incorporating information about *every* arc, rather than only about those in $T$. The parameter $\rho$ can be chosen according to the structure of the MCF problem at hand, with different values proposed in [7] for different classes of MCFs. Note that, with $\rho = 1$, $diag(M'_T) = diag(M)$; therefore, this preconditioner should be at least as effective as the diagonal preconditioner in the early stages of the IP algorithm, and comparable with the tree preconditioner in the final iterations, without the need of an explicit switch between the two. This is indirectly confirmed by a result proven in [7], which shows that the Cholesky factorization of $M'_T$ is equal to the incomplete QR factorization of $\bar{E}^T$ proposed in [8]; thus, this preconditioner is equivalent to the one proposed in the latter paper, which appears to be effective at early iterations. Note that $M'_T$ has the same nonzero pattern than $M_T$, and therefore can be factorized through visits of the tree $T$.

Let us remark here that the matrix $M = E\Theta E^T$ is rank deficient; it is well-known that the rank of $M$ is $n$ minus the number of connected components in $G$, i.e., at most $n - 1$. It is always possible to assume that $G$ is connected, as otherwise the original MCF problem can be partitioned into a set of smaller MCF subproblems, one for each of the connected components; hence, we can assune that the rank of $M$ is $n-1$. Since it is way preferable to work with full-rank matrices, a row of $E$ is usually deleted, resulting in a full-rank $(n-1) \times (n-1)$ matrix $M$. In the following, we will always assume that this has, in fact, been done. The choice of the row (node) to be eliminated is arbitrary, yet it has some implications, as discussed in Section 5.

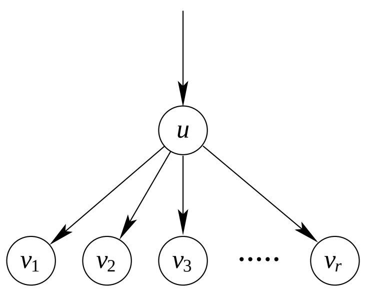
## 4. Subgraph-based preconditioners

The tree-based preconditioners have good performances when the IP methods approaches the solution, as the "large" entries of the matrix $\Theta$ naturally tend to concentrate on a spanning tree of $G$; however, they work less well at the early stages of the IP method, where "large" entries of the matrix $\Theta$ are more evenly spread among the arcs of $G$. This can be intuitively interpreted as the fact that many arcs with "large" weight are left out from the preconditioner, which therefore gives a less precise estimate of the spectrum of the original matrix. Thus, in order to improve on the performances of the tree preconditioner, especially in the early stages, we extend the family of tree-based preconditioners to a more general class of *subgraph-based* preconditioners. These will be of the form (3), where $S$ contains – possibly strictly – a spanning tree $T$.

In the choice of $S$, two contrasting goals have to be satisfied. On the one hand, $M_S$ should be a good preconditioner, and therefore $A_S$ should contain "many" arcs $(i,j)$ with "large" weights $\theta_{ij}$, as this helps in obtaining good spectral properties in the preconditioned matrix [5]. On the other hand, linear systems involving $M_S$ must be solved very efficiently (hopefully, in linear time $O(n)$), since they have to be solved at each iteration of the PCG method.

Indeed, each preconditioner in our family will be characterized by two key ingredients: one is the subgraph $S$, which dictates the spectral properties of $M_S$ and therefore its effectiveness as a preconditioner, and the other is a "good" ordering of the rows of $E_S$ (nodes of $G$), which produces a matrix $M_S$ with low fill-in Cholesky factor and therefore ensures that systems involving $M_S$ can be efficiently solved.

These two ingredients are in fact present even in the tree-based preconditioners, where $S = T$. The property that, in this case, systems with the matrices $E_S$ and $E_S^T$ can be solved in $O(n)$ can be restated in terms of the existence of a proper ordering of the rows of $E_S^T$ (nodes of $G$) such that $M_S$ exhibits no fill-in. It is important to remark that this is not true in general, i.e., fill-in can be created with a "bad" ordering of the rows. Consider for instance a tree $T$ composed by an arc entering in a node $u$ and by $r$ nodes $v_1, v_2, \ldots, v_r$ connected with $u$ by $r$ arcs (see Figure 1). If we consider the matrix $M_S$, where the rows and columns are indexed by $u$ and the nodes $v_i$



$$M_S = \begin{bmatrix} r+1 & -1 & \cdots & \cdots & -1 \\ -1 & 1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ -1 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Figure 1: A star tree $S$ and the corresponding matrix $M_S$

for $i = 1, \dots, r$ in this order, we have that $M_S$ has nonzero entries only on the diagonal and on the first row and column. However, its Cholesky factor $L$ has nonzero entries in all positions under the main diagonal: the first column of $L$ has all entries equal to $-1/(r+1)$, consequently all entries on the second column are nonzero and equal to $-1/r$, the entries on the third column are equal to $-1/(r-1)$ and so on, that is

$$L = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ -\frac{1}{r+1} & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & -\frac{1}{r} & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & -\frac{1}{r-1} & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & 0 \\ -\frac{1}{r+1} & -\frac{1}{r} & -\frac{1}{r-1} & \cdots & -\frac{1}{2} & 1 \end{bmatrix}.$$

The fact that systems with the matrices $E_S$ and $E_S^T$ can be solved in $O(n)$ is equivalent to the existence of a reordering, i.e., a $n \times n$ permutation matrix $P_n$, such that $P_n E_S$ is lower triangular; this condition it is clearly sufficient to ensure that the reordered matrix $M_S$ has a Cholesky factorization without fill-in. A suitable $P_n$ corresponds to any permutation $\mathcal{P}$ of the nodes such that if $(i, j)$ is an arc of $T$ with $i$ father of $j$, then row/column $j$ precedes row/column $i$ in $\mathcal{P}$; these permutations include reverse depth first visit and reverse breadth first visit.

Thus, tree-based preconditioners provide a matrix $E_S$ and a reordering such that $M_S$ has no fill-in. Moreover, the search for a suitable tree can be done in $O(m)$ with an approximated Kruskal algorithm, where the arc are partially ordered with the bucket sort algorithm. Note that such a search needs only to be done once at each iteration of the IP algorithm, while the solution of the system $M_S z = r$ has to be performed at each PCG iteration.

### 4.1. Brother-connected trees

We define a new family of preconditioners of the form (3), based on the characterization of a class of subgraphs $S$ (strictly containing spanning trees) for which there exists a reordering $P_n$ such that the corresponding reordered preconditioner $P_n M_S$ has no fill-in. In view of the previous analisys, these subgraphs should provide efficient preconditioners.

**Definition 4.1.** *A subgraph $S = (N, A_S)$ of $G$ is a* brother-connected tree *(BCT) if either it is a spanning tree $T = (N, A_T)$ of $G$, or it contains a spanning tree $T$ of $G$ such that the subgraph*

*$S' = (N, A_S \setminus A_T)$ obtained by removing all the arcs of $T$ from $S$ is formed of a certain number $k \geq 1$ of node-disjoint connected components $S'_1 = (N_1, A_1), \ldots, S'_k = (N_k, A_k)$ such that all the nodes in $N_i$ are "brothers" (sons of the same node) in $T$, and each $S'_i$ is a brother-connected tree.*

Definition 4.1 is inherently recoursive and operational in nature; a BCT can be constructed by iteratively taking a family of BCTs (which may be ordinary trees) and joining all their nodes in a tree where all the nodes of one of the original BCTs are sons of the same node. Note that, conversely, it is *not* required that all the sons of the same node in $T$ belong to the same connected component. In particular, the connected components can be composed by only one node; in this case, we consider the empty set of arcs to be a spanning tree (and, therefore, a BCT) for the component. In other words, the arc set $A_S$ of a BCT $S$ is the union of the arc sets of a set $\mathcal{T} = \{T_1, \ldots, T_q\}$ of arc-disjoint subtrees $T_i$ of $G$. The set $\mathcal{T}$ itself has a tree structure, where a tree $T_i$ is the son of a tree $T_j$ in $\mathcal{T}$ if all the nodes in $T_i$ are brothers in $T_j$.

Thus, an important characteristic of a BCT $S$ is its *depth*, which is the depth of the associated $\mathcal{T}$, i.e., the number of times that the composition operation has to be applied, starting from an empty graph, in order to construct $S$. A BCT of depth 1 is an ordinary tree, a BCT of depth 2 contains a spanning tree $T$ such that the removal of all the arcs in $T$ leaves a forest, and so on.

We will prove that any preconditioner in the form (3) can be factorized without fill-in if $S$ is a brother-connected tree. To do that, let us start by recalling the formulae for the Cholesky factorization of $M$:

$$\begin{aligned} d_j &= m_{jj} - \sum_{k=1}^{j-1} (l_{jk})^2 d_k && \text{for } j = 1, \ldots, n \\ l_{ij} &= 1/d_j [m_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} d_k] && \text{for } i = 1, \ldots, j-1, j = 1, \ldots, n \end{aligned}$$

where $m_{ij}$ and $l_{ij}$ are the entries of the matrices $M$ and $L$, respectively, while $d_j$ are the diagonal entries of $D$.

**Lemma 4.1.** *Let $M'$ be a matrix with Cholesky factor $L'$. Any symmetric positive definite matrix*

$$M = \begin{bmatrix} M' & m \\ m^T & \mu \end{bmatrix}$$

*has a Cholesky factor of the form*

$$L = \begin{bmatrix} L' & 0 \\ l^T & 1 \end{bmatrix}.$$

This well-known property derives directly from the formulae of Cholesky factorization. Note that the values in a row of the Cholesky factor $L$ depends only on the values on the same row and on the previous ones. Therefore, if $M'$ is a matrix with no fill-in, then $M$ can only have fill-in in its final row.

The structure of the nonzeroes of the matrices $M = E\Theta E^T$ that we are interested in is completely determined by the topological structure of the underlying graph $G$; it is interesting to note that, conversely, to any symmetric matrix $A$ we can associate an undirected graph $G_A$ which describes its nonzero structure. The nodes of $G_A$ are associated with the rows/columns of $A$, while its edges are associated with the non zero entries of $A$, i.e., the edge $\{i, j\}$ exists if and only if $a_{ij} \neq 0$. This double-sided link between matrices and graphs can be profitably exploited for theoretical studies in both fields [4, 2, 5]. As in network flow problems the graph $G_M$ is the undirected version of the directed graph $G$ on which the problem is defined, we usually use the term arc instead of the term edge.

In graph theoretical terms, the operation described in Lemma 4.1 corresponds to the addition of a new node to the graph $G'_M$, possibly connected with all other nodes. Therefore, the following result easily follows from Lemma 4.1.

**Lemma 4.2.** *Consider any finite number $k \geq 1$ of node-disjoint graphs $G_i = (N_i, A_i)$ for which there exists a proper ordering $\mathcal{P}_i$ of the node sets $N_i$ such that the matrix $M_{G_i}$ can be factorized without fill-in, for $i = 1 \ldots k$. The graph $G = (N, A)$ obtained as the union of all the graphs $G_i$ plus a new node $u$ linked by an arc to each node in each of the graphs $G_i$ has the same property, i.e., there exists a proper ordering $\mathcal{P}$ of the nodes set $N$ such that the matrix $M_G$ can be factorized without fill-in.*

*Proof:* The desired permutation $\mathcal{P}$ is easily obtained by composing the permutations relative to each $G_i$, where the order of the different $G_i$ is arbitrary, and placing the new node $u$ as the last node in the ordering. In this way, the $(n-1) \times (n-1)$ principal submatrix $M'$ of (the reordered) $M_G$ can be factorized without fill-in. Now we can apply Lemma 4.1 and the subsequent observations, where $u$ corresponds to the new row/column $[m^T \mu]$, to show that even $M = M_G$ can be factorized without fill-in. In fact, the new row $l$ in the Cholesky factor of $M$ is dense (completely nonzero), but this corresponds to the fact that $S$ has $n-1$ arcs more than $S'$, i.e., the row $[m^T \mu]$ is completely nonzero too. $\square$

We can now prove the following fundamental theorem.

**Theorem 4.3.** *Given a brother-connected tree $S$ in $G$, there exists an ordering of the nodes of $G$ such that the preconditioner $M_S$ has a Cholesky factor $L$ with no fill-in.*

*Proof:* We will proceed by double nested induction: the first on the depth of $S$, the second on the number of non terminal nodes in the tree $T$ contained in a brother-connected tree.

The base step of the (outer) induction corresponds to depth 1, i.e., to the case where $S$ is a tree. In this case, the result is well-known and it has already been recalled: any ordering of the nodes such that node $j$ precedes node $i$ if $(i, j)$ is an arc of $S$ and $i$ is the father of $j$ has the desired property.

For the inductive step, we will assume that for every BCT with depth at most $h$ a proper reordering of the nodes can be found such that the corresponding matrix can be factorized without fill-in, and we will prove that the property holds for BCTs of depth $h+1$. Once again we proceed by induction, this time on the number of non terminal nodes of the spanning tree $T$ contained in $S$.

The base step of the induction corresponds to the case where there is only one non terminal node $u$, i.e., $T$ is a "star tree" where any other node but $u$ is a leaf. In this case, consider the subgraph $S'$ obtained by removing $u$ (and all its incident arcs) from $S$; since $S$ is a brother-connected tree, $S'$ is formed of $k \geq 1$ node-disjoint brother-connected trees *of depth at most $h$*. Therefore, for the (outer) inductive hypothesis, for each of the components there exists an ordering such that the corresponding matrix can be factorized without fill-in, and we can apply Lemma 4.2.

For the (inner) inductive step, consider a non terminal node $u$ such that all its sons $v_1, v_2, \ldots, v_r$ are leaves of $T$. Let $S'$ be the subgraph of $S$ induced by the nodes $V = \{v_1, \ldots, v_r\}$, and let $S''$ be the subgraph of $S$ induced by all the other nodes (the arcs going from $u$ to the nodes in $V$ belong neither to $S'$ nor to $S''$).

Now, we can apply the (inner) inductive hypothesis to $S''$, as we have reduced the number of non terminal nodes by one unit; hence, the matrix $M''_S$ can be factorized without fill-in. The same is true for the matrix $M'_S$, since – as in the base step of the (inner) induction – $S$ is a brother-connected tree, hence $S'$ is a set of node-disjoint BCTs of depth at most $h$.

Now, consider the two orderings $\mathcal{P}'$ and $\mathcal{P}''$ of the nodes in $V$ and $N \setminus V$, respectively, which allow to factorize $M'_S$ and $M''_S$ without fill-in. Construct an ordering $\mathcal{P}$ of $N$ that corresponds to $\mathcal{P}'$ on $V$, to $\mathcal{P}''$ on $N \setminus V$ and such that all the nodes in $V$ precede those in $N \setminus V$. Since $u$ is a "leaf" in $S''$, we can assume without loss of generality that it is the first node in the ordering $\mathcal{P}''$. Therefore, the matrix $M_S$ corresponding to the reordering $\mathcal{P}$ has the following form:

$$\left[\begin{array}{c|c} M'_S & m \mid 0 \\ \hline \begin{array}{c} m^T \\ \hline 0 \end{array} & M''_S \end{array}\right]$$

Hence, the first part of the Cholesky factor $L$ of $M_S$ is equal to that $M'_S$, and all the entries $l_{ij}$ for $j \leq r$ and $i > r+1$ are zero; the only possible nonzeroes can be created in the row $r+1$, the one corresponding to the node $u$. However, as in Lemma 4.2, these $r$ new nonzeroes exactly corresponds to the $r$ arcs joining $u$ with all its sons, which had been left out from both $S'$ and $S''$. Thus, from Lemma 4.1 we know that the Cholesky factorization of the $(r+1) \times (r+1)$ principal submatrix of (the reordered) $M_S$ has no fill-in, so consider the diagonal entry in row $r+1$ of $D$, denoted by $d_u$, corresponding to node $u$. From the formulae of the Cholesky factorization, it is easy to see that the rest of the factorization of $M_S$ can be computed exactly as the factorization of $M''_S$, only starting from the value $d_u$ instead of the original value $m_{uu}$. For inductive hypothesis, we know that the Cholesky factor of $M''_S$ (reordered with $\mathcal{P}''$) has no fill-in, and this finally allows us to conclude that the Cholesky factor of the whole (reordered) $M_S$ has no fill-in. □

Thus, every BCT results in an efficient preconditioner, provided that che corresponding "good" reordering $\mathcal{P}$ can be found. For instance, for a BCT with depth two $\mathcal{P}$ need only be such that:

- ⋄ the matrix $E_T$ associated with its spanning tree $T$ is lower triangular, i.e., if $(i, j)$ is an arc of $T$ with $i$ father of $j$, then row/column $j$ precedes row/column $i$ in $\mathcal{P}$;
- ⋄ for each non terminal node $u$ of $T$, each subset of its sons which belong to the same subtree $T_h = S'_h$ (once that the arcs of $T$ have been removed) are ordered in the permutation according to the order defined by $T_h$, i.e., if $(i, j)$ is an arc of $T_h$ with $i$ father of $j$, then row/column $j$ precedes row/column $i$ in $\mathcal{P}$; the roots of the subtrees and the sequence of the trees can be arbitrarly chosen.

In general, a "good" permutation can be recursively constructed by ordering the nodes of the BCTs of depth $h$, and then composing these orders into orders for the BCTs of depth $h+1$. This can be done with a bottom-up postvisit of the tree $\mathcal{T}$ associated with the BCT, i.e., by visiting the tree $\mathcal{T}$ from the leaves to the root with the constraint that each node of $\mathcal{T}$ can be visited only after all its sons. Note, however, that this information is "richer" than a description of a BCT simply as a list of arcs; in principle, extracting $\mathcal{T}$ from a BCT is nontrivial.

The induction process in the proof of Theorem 4.3 can be used to construct an algorithm for performing the Cholesky factorization of $M_S$ without fill-in in $O(nh^2)$, where $h$ is the depth of the BCT. Note that all the trees at the same depth $q$ can be represented with an unique predecessor function $Pred[q]$ defined on the nodes, such that $Pred[q][u] = v$ if $v$ is father of $u$ at depth $q$, and $Pred[q][u] = nil$ if $u$ is a root, i.e., it has no father. For instance, in a BCT of depth two the function $Pred[1]$ represents the spanning tree $T$ whose removal leaves a forest $F$, which is represented by the function $Pred[2]$. The algorithm for computing the $LDL^T$ factorization of a matrix $M_S$ is shown in the following pseudo-code; it requires a description of the BCT $S$ of depth $h$ on a graph with $n$ nodes in terms of the predecessor functions $Pred[q]$, and a description of a "good" ordering $\mathcal{P}$ in an array $Order[]$. The algorithm outputs tqhe Cholesky factor $L$ and the diagonal matrix $D$.

```
Procedure $CholeskyFactorBCT(h, n, M, Pred, Order, L, D)$
  begin
    for $i = j, \dots, n; i = 1, \dots, j-1$ do
      $L[i,j] = M[i,j]$;
    for $i = 1 \dots n$ do
      $D[i] = M[i,i]$;
      $L[i,i] = 1$;
    for $i = 1 \dots n-1$ do
      $u = Order[i]$;
      for $q = h \dots 1$ do
        $w = Pred[q][u]$;
        if $w \neq nil$ then
          $L[w,u] = L[w,u]/D[u]$;
          $D[w] = D[w] - L[w,u]^2 * D[u]$;
      for $q = h \dots 1$ do
        $w = Pred[q][u]$;
        if $w \neq nil$ then
          for $r = q-1 \dots 1$ do
            $y = Pred[r][w]$;
            if $y \neq nil$ then
              $L[y,w] = L[y,w] - L[w,u] * L[y,u] * D[u]$;
  end.
```

Using the same data structures, $Pred$ and $Order$, an $O(nh)$ algorithm that solves systems of the form $M_S r = v$ – which is what is actually required if $M_S$ is used as a preconditioner in a PCG algorithm – can be constructed. Thus, any iteration of a PCG algorithm which uses a subgraph-based preconditioner where $S$ is a BCT of depth $h$ has a complexity of $O(nh + m)$.

In our implementation, we have only considered the case of BCTs of depth two. This simplifies and streamlines the algorithms, while still leaving room for almost doubling the number of arcs to be put in $S$ with respect to a standard tree-based preconditioner (a BCT of depth two can have up to $2n - 3$ arcs).

## 5. Finding brother-connected trees

From the results in the previous section, we know that any brother-connected tree $S$ of level two provides a preconditioner $M_S$ which is essentially as expensive as a tree-based preconditioner, but – hopefully – more effective. Thus, the problem arises of finding the "best" BCT. As in the case of tree-based preconditioners, it is reasonable to assume that a brother-connected tree with large total weight (using $\theta_{ij}$ as the weight of arc $(i,j)$) should provide a good preconditioner. Unfortunately, the complexity of the problem of finding the maximum-weight brother-connected tree in a given graph $G$ is not known to us. However, the exact solution of this problem is not crucial for its application in the PCG algorithm. In fact, the preconditioners have to be found quickly, possibly in linear time, in order for the whole process to be competitive. It is very unlikely that a(n approximate) maximum-weight BCT can be found in linear time, as is done for maximum-weight spanning tree, because BCTs are not matroids. This is shown by the two BCTs $S_1$ and $S_2$ in Figure 2, where the solid arcs belong to the first level, the dashed ones belong to the second level, and, finally, the dotted ones are in the complements $G \setminus S_1$ and $G \setminus S_2$. It is easy to check that $S_1$ and $S_2$ are maximal BCTs (of level two) with different cardinality.

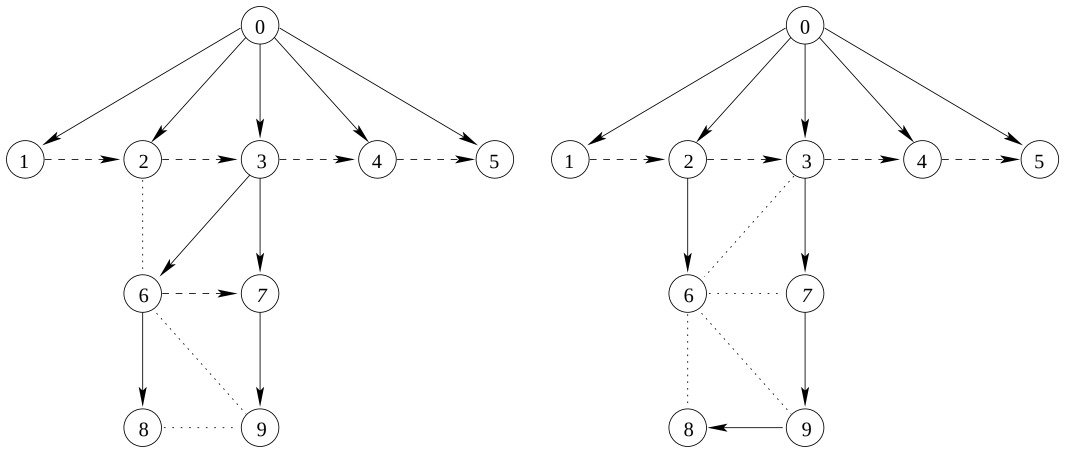


Figure 2: Two maximal BCTs on the same graph with different cardinality

Furthermore, there is no proof that the maximum-weight BCT provides the largest decrease in the condition number of the matrix, although preconditioners with many entries equal (or close) to those of the original matrix are usually found to effectively reduce the condition number (in fact, the best theoretical preconditioner is the matrix $M$ itself). Finally, the condition number only gives an upper bound on the number of PCG iterations, so that the BCT preconditioner which most reduces the condition number may still not be the best BCT preconditioner in practice.

For all the above reasons, we will resort to heuristics to find the BCT to be applied in the PCG method. A large number of different heuristics can be proposed, by combining different variants of two basic ingredients:

i) how a spanning tree $T$ is chosen;

ii) how extra arcs forming trees among brothers (sons of the same node) in $T$ can be chosen.

### 5.1. Finding an initial spanning tree

For point (i), we propose three different variants:

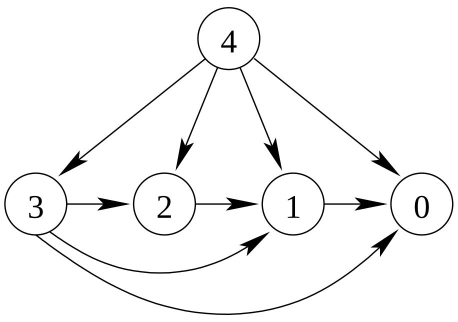
i.a) An approximate maximum-weight spanning tree $T$ is constructed in $O(m)$ with the Kruskal algorithm applied to an approximated ordering of the arcs (using a “bucket” structure); this is precisely the approach proposed in [12, 10] for finding the tree preconditioner.

i.b) The tree is constructed with a simple breadth-first visit starting from one designated root node; this hopefully produces a “shallow” tree where the nodes have “many” sons, thus providing the chance of larger trees in the second level.

i.c) Mixing the above two ideas, the tree is constructed with a breadth-first visit which tries to use first arcs with a large weight. This can be done by dividing the arcs into $k$ classes, according to their weight, and defining a queue for each class of arcs. Starting from the root, any new visited node is inserted in the first queue (the one associated with arcs with the largest weights). When a node is extracted from a queue, it is moved in the queue corresponding to the next class of arcs (if any), and then all the arcs in the class associated with this queue are visited. The total complexity is $O(nk)$, which can be considered linear if $k$ is a small fixed value (e.g., $k = 3$).

For each of the above three techniques, we also propose a variant which handles the root node in a different way. The idea is to obtain a tree with “few” arcs incident in the root node. The rationale for such a choice can be found in the following example.

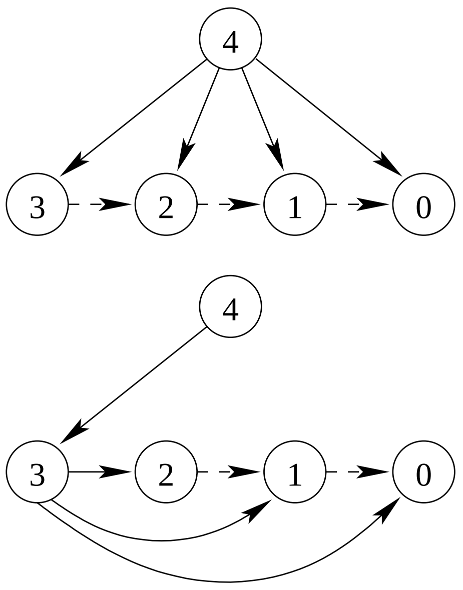
**Example 5.1.** Consider the graph $G$ in Figure 3 and the corresponding matrix $M = EE^T$. Note



$$\begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 3 & -1 & 0 \\ -1 & -1 & 4 & -1 \\ -1 & 0 & -1 & 3 \end{bmatrix}$$

Figure 3: The graph $G$ and the associated matrix $M$

that the row of $E$, corresponding to the “root node” 4, has been deleted in order to make $E$ full-rank. Now, consider the two brother-connected trees in Figure 4. Although the BCT on the left has more edges than the one on the right, it is the latter who gives a preconditioner with more nonzero entries; indeed, this preconditioner differs from the original $M$ only in the diagonal elements (this is particularly interesting in view of the improvements suggested in Section 6, as differences in the diagonal elements are easily recovered). This is due to the fact that the weight of an arc incident in the root contributes only to one diagonal entry of the preconditioner, while the weight of an arc not incident in the root contributes to both a diagonal and a nondiagonal entry of the preconditioner. Thus, arcs incident in the root “contribute less” to the preconditioner than arcs not incident in the root with the same weight. □



$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

Figure 4: Two BCTs; dashed arcs correspond to the second level

Note that it is customary to associate the root node of the tree $T$ with the node that is deleted, and we have also followed this convention, although in principle different choices would be possible. In order to take into account the behaviour highlighted in the example, we have implemented variants of (i.a), (i.b), and (i.c) that attempt to construct a tree with as few arcs incident in the root node as possible (at least one is clearly necessary). This can be easily done by applying each procedure to the subgraph obtained by canceling the root node, and then adding the necessary arcs incident in the root (one for each of the disconnected components created in the graph by the removal of the root node). It is easy to see that this can be done without affecting the worst-case complexity of the original algorithms.

Finally, we consider two possible strategies to select the root node: with the first ("static") strategy, the node with the largest adjacency list is selected, while with the second ("dynamic") strategy, the node with the largest total weight $\sum_{e\in\delta^{\pm}(u)}\theta_e$ of the set of incident arcs is selected. The rationale behind these two rules is to obtain a matrix $M$ which is the most *diagonally dominant*, and therefore that is better suited for a PCG method. A matrix is column (row) diagonally dominant, if, for each column (row), the sum of the modules of the entries out of the diagonal is less than or equal to the module of the diagonal entry, and strict inequality holds for at least one column (row). Diagonal dominance has effect on the range of the eigenvalues of the resulting matrices, as shown, for instance, by the well-known Gerschgoring's theorems, and therefore on the spectral conditioning of the matrix. It is easy to see that, if $G$ is connected, the matrix $M = E\Theta E^T$ – where any row has been deleted from $E$ – is column (and row) diagonally dominant. Moreover, each non-zero entry out of the diagonal on the row that is canceled corresponds to a column/row that respects diagonal dominance rule with strictly inequality. These nonzero entries exactly correspond to arcs incident in the deleted node, that is taken to be the root of the tree. The larger the weights of these arcs, the larger these entries are, and therefore the more the matrix is diagonally dominant. The dynamic rule typically gives better results, but it requires to select the node at each IP iteration.

## 5.2. Enlarging the tree to a BCT

For point (ii), i.e., how extra arcs forming trees among brothers in $T$ can be chosen, we propose three different variants:

ii.a) When the tree is selected, the final ordering of the nodes to be considered in the factorization is also arbitrarly fixed. Then, the arcs out of $T$ are scanned in (approximated) order of decreasing weight and added to the tree if they do not affect the permutation decided in the first step, and they satisfy the condition that the trees on the second level are paths among brothers.

ii.b) The trees in the second level of the BCT are restricted to be paths. As in case (ii.a), the arcs are scanned in (approximated) order. The final ordering is found by considering one the two possible permutations for each path among brother nodes, and then composing these orders respecting the tree structure of $T$.

ii.c) Analogous to case (ii.b), but the trees in the second level of the BCT are not restricted to be paths.

These three variants require a different amount of data structures and computation time (how many times the list of arcs is scanned), and they find different BCTs. Variant (ii.a) is the cheapest one, but it usually adds less new arcs. On the other hand, Variant (ii.c) is the most complex, as it requires a new union-find structure to find trees in the second level and to select the order taking into account these trees, but it is usually more successfull in adding a fair number of new arcs to the BCT. Variant (ii.b) is something in between.

For variants (ii.b) and (ii.c), it is actually possible to modify the original spanning tree $T$ as the algorithm proceeds, in order to allow the addiction of arcs that would otherwise be impossible to add. One way for doing that is to apply an operation, which we call *promotion*, whereby a node connected with its grandfather is "promoted" as a brother of its former father. More in detail, let $j$ be a node, $k$ its father in $T$, and $i$ the father of $k$ in $T$. If the arc $(i, j)$ is selected from the (approximated) ordering, it is possible, under some conditions, to modify the tree $T$ in such a way that $j$ becomes son of $i$ and brother of $k$. In this way the arc $(i, j)$ is added as an arc of $T$ (i.e., in the first level of the BCT), and the arc $(k, j)$ becomes an arc of the second level (see Figure 5). The main condition to apply the promotion is that node $j$ must not have incident arcs $(j, l)$ in the second level, as after the promotion $j$ and $l$ are no longer brothers. Moreover, for variant (ii.b), the node $k$ is required to have at most one connected brother in the

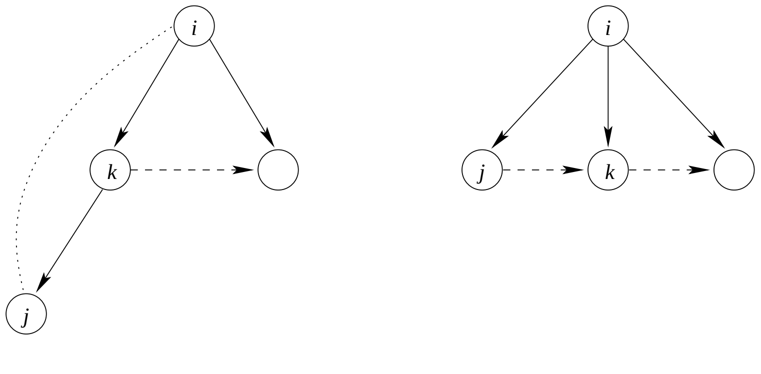


Figure 5: The BCT before (on the left) and after (on the right) the promotion

second level of the BCT, otherwise the constraint that the trees in the second level are paths would no longer be satisfied.

## 6. Further improvements

All the preconditioners that we have proposed so far can be further improved by applying two kinds of operations that attempt to incorporate in $M_S$ information regarding arcs which have been left out of the subgraph $S$.

The first operation amounts at adding to $S$ all arcs $(i, j)$ which are "parallel" to arcs already belonging to $S$, i.e., every other arc $(i, j)$ or $(j, i)$ belonging to $G$. Clearly, this cannot generate fill-in other than that already present in the original $M_S$, as the support of the two matrices is the same. Note that "parallel" arcs, i.e., multiple copies of the same arc (or of its reverse arc) with different costs and capacities, are often found in MCF problems, e.g. to model piecewise linear convex separable flow cost functions [1]. Surprisingly, this kind of operation does not seem to have been proposed before in the literature.

The second operation, proposed in [7] for the tree-based preconditioner, consists, given a preconditioner $M_S$, in using as "final" preconditioner the matrix

$$M'_S = M_S + \rho \; diag(M - M_S) \; ,$$

where $diag(A)$ is the diagonal matrix having as the diagonal elements those of $A$. The matrix $M'_S$ cannot have more fill-in than $M_S$, and it at least contains some information about *every* arc.

This second operation ties in well with the variants that attempt to construct BCTs with few arcs exiting from the deleted node (cfr. Section 5.1). In fact, the rationale behind those variants is that these arcs contribute only to the diagonal entries of the preconditioner, and this contribution is completely recovered by the addition *a posteriori* of $diag(M - M_S)$ (i.e., $\rho = 1$). In fact, consider once again Example 5.1; if we adopt the preconditioner $M'_S$ starting from the second BCT of Figure 4, we obtain that $M'_S$ is equal to $M$, so it is an optimal preconditioner. This is not true if the other BCT is used instead, i.e., if more arcs incident in the root are selected, thereby discarding arcs that are not incident in the root.

## 7. Conclusion

We have proposed a new family of preconditioners for the solution of the "core" system of equations arising in the solution of Min Cost Flow problems through an Interior Point method. These preconditioners improve on those known in the literature by giving a larger flexibility in how to select the subgraph, and therefore allowing a finer tuning of the trade-off between the cost of computing and using the preconditioner and the corresponding reduction in the number of Preconditioned Conjugate Gradient iterations. These preconditioners are based on a – to the best of our knowledge – new family of graphs, the study of whose properties may prove interesting in itself. In a companion paper we will present the results of a large-scale computational experience aimed at assessing the effectiveness of the different variants of preconditioners for several classes of MCF problems and many combinations of different IP algorithms.

Further work along this line of research will involve testing the resulting MCF algorithm against efficient MCF codes from the literature. Also, the effectiveness of BCT-based preconditioners using BCTs of depth larger than 2 will have to be tested. Finally, other preconditioners could be constructed by relaxing the condition that their final Cholesky factor should have no fill-in.

## References

[1] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: theory, algorithms and applications.* Prentice Hall, New Jersey, 1993.

[2] W. Anderson and T. Morley, "Eigenvalues of the Laplacian of a graph," *Linear and Multilin. Algebra*, vol. 18, pp. 141–145, 1985.

[3] J. Castro, "A specialized interior-point algorithm for multicommodity network flows," *SIAM Journal on Optimization*, vol. 10, pp. 852–877, 2000.

[4] D. Cvetkovic, M. Doob, and H. Sachs, *Spectra of Graphs.* Academic Press, New York, 1979.

[5] A. Frangioni and S. Serra Capizzano, "Spectral Analysis of (Sequences of) Graph Matrices," *SIAM J. on Matrix An. and Appl.*, to appear (2001).

[6] N. K. Karmarkar and K. G. Ramakrishnan, "Private Comunication," 1988.

[7] 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.*, SIAM, 1995.

[8] L. F. Portugal, F. Bastos, J. J. Jùdice, J. Paixao, and T. Terlaky, "An investigation of interior point algorithms for the linear transportation problem.," *SIAM Journal on Computing*, no. 17, pp. 1202–1223, 1996.

[9] L. F. Portugal, M. G. C. Resende, G. Veiga, and J. J. Jùdice, "An efficient implementation of an infeasible primal-dual network flow method," tech. rep., AT&T Bell Laboratories, Murray Hill, New Jersey, 1994.

[10] L. F. Portugal, M. G. C. Resende, G. Veiga, and J. J. Jùdice, "A Truncated Primal-infeasible Dual-feasible Network Interior Point Method," *Networks*, vol. 35, pp. 91–108, 2000.

[11] M. Resende and G. Veiga, "An efficient implementation of a network interior point method," *Technical report, AT&T Bell Laboratories, Murray Hill, NJ.*, 1992.

[12] M. Resende and G. Veiga, "An Implementation of the dual affine scaling algorithm for minumum cost flow on bipartite uncapacited networks," *SIAM Journal on Optimization*, vol. 3/3, pp. 516–537, 1993.

[13] T. Roos, T. Terlaky, and J.-P. Vial, *Theory and Algorithms for Linear Optimization: An Interior Point Approach.* John Wiley and Sons, Chichester, 1997.

[14] D. J. Rose and R. E. Tarjan, "Algorithmic aspects of vertex elimination on graphs," in *7th Annual Symposium on the Theory of Computing, Association for Computing Machinery*, (New York), pp. 245–254, 1975.

[15] P. Vaidya, *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners.* Urbana, IL: Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1990.

[16] S. Wright, *Primal-Dual Interior-Point Methods.* SIAM, Philadelphia, PA, 1997.