Accelerated multigrid for graph Laplacian operators

Pietro Dell'Acqua[a], Antonio Frangioni[b], Stefano Serra-Capizzano[a,c]

[a] *Dipartimento di Scienza e Alta Tecnologia, Università dell'Insubria, Via Valleggio 11, 22100 Como, Italy.*
[b] *Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy.*
[c] *Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden.*

Abstract

We consider multigrid type techniques for the numerical solution of large linear systems, whose coefficient matrices show the structure of (weighted) graph Laplacian operators. We combine ad hoc coarser-grid operators with iterative techniques used as smoothers. Empirical tests suggest that the most effective smoothers have to be of Krylov type with subgraph preconditioners, while the projectors, which define the coarser-grid operators, have to be designed for maintaining as much as possible the graph structure of the projected matrix at the inner levels. The main theoretical contribution of the paper is the characterization of necessary and sufficient conditions for preserving the graph structure. In this framework it is possible to explain why the classical projectors inherited from differential equations are good in the differential context and why they may behave unsatisfactorily for unstructured graphs. Furthermore, we report and discuss several numerical experiments, showing that our approach is effective even in very difficult cases where the known approaches are rather slow. As a conclusion, the main advantage of the proposed approach is the robustness, since our multigrid type technique behaves uniformly well in all cases, without requiring either the setting or the knowledge of critical parameters, as it happens when using the best known preconditioned Krylov methods.

Keywords: graph matrices, multigrid, conditioning and preconditioning
2000 MSC: 05C50, 15A12, 65F10

1. Introduction

The paper is concerned with solution methods for (extremely) large linear systems whose matrices have a graph structure with weighing on the arcs (see, e.g.,[19] and references therein). We concentrate our efforts in particular on the symmetric case, with matrices arising from graph *Laplacian* operators. Such a type of matrices are encountered in several applications where a network structure is present, ranging from Web searching engines [46] to general Markov chains [20], from consensus algorithms [53] to optimization problems such as the Minimum Cost Flow (MCF) in networks [1, 4]. In all these cases the dimension of the involved matrices is either very large or extremely large. Among these applications, the central one in the development of this article, at least as far as the computational part is concerned, involves linear systems arising in Interior Point (IP) techniques for MCF problems [1, 4]. In this setting, a linear system involving the weighted Laplacian of the underlying network has to be solved at each iteration, with varying vector of arc weights Θ (cf. relation (2)). Not only the networks can be very large (e.g., with up to 2^{22} arcs [54]), but these approaches allow a more or less direct extension [13, 14] to *multicommodity* flow problems [30, 15, 16], that have a huge range of practical applications from telecommunication [18] to transportation [31] and beyond. In the latter setting, the size of the matrix is further multiplied by the number of *commodities* (different types of flows in the network), that can be easily run into the thousands (e.g. being *quadratic* in the number of nodes in some applications). Furthermore, the graph structure is somewhat "muddled" by the rows corresponding to mutual capacity constraints.

In summary, the considered class of problems exhibits three main issues:

- the large size of the considered linear systems;
- the sparsity and the graph structure of the involved matrices;

Email addresses: pietro.dellacqua@gmail.com (Pietro Dell'Acqua), frangio@di.unipi.it (Antonio Frangioni), s.serracapizzano@uninsubria.it (Stefano Serra-Capizzano)

- the potential ill-conditioning, as a function of the matrix dimension and/or of other critical parameters (such as the weights vector Θ).

These features should immediately refrain from using direct solvers. In fact, while iterative methods exploit the structure and the sparsity since each iteration is in general reduced to matrix-vector products, the methods based on factorizations [36], such as Gaussian LR, QR or real Cholesky LL^T (only for the MCF problem), in general do not exploit naturally the structure and in particular the sparsity of the matrices [13] and they need special techniques such as elimination trees, ordering etc. in order to achieve an acceptable cost, both in space and time. However, for very large dimensions and in the case of ill-conditioning, the presence of amplified round-off errors is the main difficulty encountered by direct methods.

The use of iterative solvers has the immediate advantage that the matrix vector product can be done within linear work in case of sparsity and its stability is guaranteed in the sense that only a moderate amplification of the round-off errors occurs. Moreover no storage problems arise since the original coefficient matrix is not manipulated. However, the advantages in the reduced cost per iteration could be negated by a huge number of iterations, i.e., slow convergence, stagnation, non convergence. Finding fast convergent methods is non-trivial (see [47] for a very interesting paper dealing with multigrid techniques for graph matrices), and essentially it relies on spectral information, which may be difficult to identity. Ideally, one would like to design *optimal* methods in which the number of iterations for reaching a pre-assigned accuracy $\varepsilon > 0$ is independent of the dimension and (possibly) of the other parameters involved in the problem, and only depends on the parameter ε (for a precise notion of optimality see [3]). This difficulty is typically related to ill-conditioning: in our setting of positive (semi) definite matrices, the latter is related to the ratio between the largest and the smallest non-zero eigenvalue, while in the general setting where non Hermitian matrices are involved, not only the extremes of the spectrum, but also other measures are relevant, such as the non-normality measure of the matrix [5] or the ill-conditioning of the eigenvector matrix. In fact, if the matrix is well-conditioned, then simple scaled Richardson methods or scaled Richardson methods for normal equations are optimal. The same can be claimed for Krylov methods such as conjugate gradient (CG) or GMRES. However, for a matrix-sequence A_n whose condition number $g(n)$ diverges as $n \to \infty$, the classical iterative solvers (relaxed Richardson, relaxed Jacobi, relaxed Gauss-Seidel, CG etc.) are not convergent or, at best, the iteration count, as a function of n, diverges as well in a way that can be related with $g(n)$ [36, 56]. As an example, we consider the discrete Laplacian Δ_n related to the equi-spaced finite difference approximation of $-d^2/dx^2$ on $(0, 1)$ with homogeneous Dirichlet boundary conditions, which can also be viewed as a MCF matrix $E\Theta E^T$ where Θ is the identity and E is the node-arc incidence matrix of a linear graph, and the discrete bi-Laplacian Δ'_n related to the same type of approximation of the operator d^4/dx^4 with homogeneous boundary conditions [40]. Since $g(n) \sim n^2$ for Δ_n and $g(n) \sim n^4$ for Δ'_n, it is clear that the speed of convergence may not be satisfactory. The following table reports the iteration count for different classical methods and for reaching a given accuracy.

	Jacobi/Richardson	Gauss-Seidel	CG
Δ_n	$O(n^2)$	$O(n^2)$	$O(n)$
Δ'_n	$O(n^4)$	$O(n^4)$	$O(n)$

A more complete analysis of the conditioning and extremal eigenvalues of more general graph matrices can be found in [35].

Three classes of approaches are known to be successful for ill-conditioned systems:

- Krylov methods with preconditioning [56];
- multigrid methods (or multi-level methods, aggregation-disaggregation, etc.) [61];
- multi-iterative solvers [57].

In the case of graph matrices, the first approach has been considered mainly through *support-graph preconditioners*. Proposed by P. Vaidya in [62], but not analyzed or implemented at that time, these are based on the idea of extracting a properly structured subgraph (such as a tree or tree-like structure) and using the corresponding matrix as preconditioner. Several authors have analyzed these preconditioners both from the theoretical and from the practical viewpoint (cf. [8, 9, 6, 49, 32, 33] and the references therein). *Support theory* was developed [9] for the study of *combinatorial subgraph preconditioners*, which lies at the heart of impressive theoretical

results culminating first in the proof that *symmetric diagonally dominant* (SDD) systems can be solved in nearly optimal $O(m \log^{O(1)} n)$ time [59], and later that SDD matrices with planar connection topologies can be solved asymptotically optimally in $O(n)$ time [44]. The usefulness in practice of these theoretical results is not completely clear, since some of the hidden constants may be very large. Another relevant avenue of research is that of *Steiner preconditioners*, introduced in [37] and extended in [43], whereby the original graph is augmented with "fake" nodes, thereby yielding a larger equivalent system with "do not care equations", that are purposely constructed to increase the effectiveness of support-graph preconditioners. In some cases Steiner trees are much better than subtrees: for instance, the condition number of the best tree for an unweighted square grid is $O(n)$, while the condition number of the best Steiner tree is $O(\sqrt{n})$. This extends to more complicated graphs: for instance, the best condition number that can be obtained for 2D grids using subgraphs with $n + n/k$ arcs is of order $O(k)$, while it is $O(\sqrt{k})$ with Steiner graphs of the same size, and the gap between the two approaches is even wider for 3D grids. Yet, more recent analysis shows that the results initially obtained for preconditioners can be easily applied to *aggregation operators* (cf. (8) for multigrid-type approaches). This gives rise e.g. to the *combinatorial multi-grid* (CMG) solver of [45], a variant of the multigrid approach running a more complex cycle than the standard V and W ones, that is then used as a preconditioner for PCG with excellent results. Such a development is in line with a recent trend of research which aims at increasing the robustness of multigrid approaches by using them not as a stand-alone solver, but rather in combination with acceleration methods such as conjugate gradient [23, 24], in accordance with the multi-iterative idea [57]. The latter is driven by the observation that it is often not only simpler, but also more efficient to use accelerated multigrid approaches rather than to try to optimize the interplay between the various multigrid components in order to improve the convergence of stand-alone multigrid cycles. Indeed, multi-iterative solvers are the combination of different iterations, where the key notion is that of *spectral complementarity*. In other words, each method could be slowly convergent (or even non-convergent) in itself, but their combination is fast since each of them substantially reduces the error in a given subspace and the global direct sum of such subspaces coincides with the whole space. Multigrid methods belong to this class since, in their simplest versions, they can be seen as the combination of a smoother, usually slowly convergent in the ill-conditioned subspace (that is related, after scaling, to the eigen-spaces associated with small eigenvalues) but very fast in the complementary one, and of a coarse-grid procedure which is not convergent at all, but strongly contracts the error in the ill-conditioned subspace. The product of these two basic iteration matrices has a spectral radius that is much small than the product of the spectral radii of the basic components and, in this sense, the multigrid procedure is a multi-iterative solver: in [57], a further *intermediate iteration* was introduced, usually as post-smoother, in order to reduce the error in the subspace where the cumulative effect of the pre-smoother and of the coarse-grid correction was less effective.

This paper follows along these research lines, with a twofold objective. From the theoretical viewpoint, we aim at characterizing the class of projectors that allow to maintain the graph structure of the matrix at all steps of the multigrid chain, so as to be able to use efficient combinatorial preconditioning techniques at all steps. From the experimental viewpoint, we aim at comparing PCG approaches and multigrid methods on some classes of linear systems which can be anything between very well-conditioned and extremely ill-conditioned; actually, several of these systems need to be solved within a given application (solution of a MCF problem via IP methods), with the conditioning varying wildly between the initial and the final systems. It is well-known than no single preconditioned Krylov method is sufficiently effective throughout, as approaches that are appropriate at one extreme become ineffective at the other, and vice-versa. In this context the multigrid-type approach proposed in this paper proves to be "robust" for all the systems, independently from their conditioning, which is a relevant feature.

The paper is organized as follows. In Section 2 we introduce in more detail the MCF problem and we describe in general terms the multigrid procedure. Section 3 is devoted to the theoretical characterization of the family of projectors that preserve the graph structure at all steps of the multigrid chain, thereby allowing combined use of (possibly, accelerated) multigrid and combinatorial preconditioning. Section 4 is devoted to the presentation of several classes of projectors with the right properties, either newly proposed or drawn from the relevant literature. Section 5 is devoted to the presentation of a large set of numerical experiments comparing preconditioned Krylov methods and accelerated multigrid methods in the context of the solution of MCF problems: in this respect see also the numerics in [26]. Finally, Section 6 reports our conclusions.

2. Preliminary steps

In this section we first present in some detail the MCF problem, which will serve as our primary benchmark for numerical tests, then we introduce the multigrid procedure to be specialized in the next sections in our graph setting.

2.1. The Minimum Cost Flow Problem

We start by recalling the definition of *node-arc* incidence matrix of a directed graph.

Definition 1. Let $\mathcal{G} \equiv \mathcal{G}_n = (\mathcal{U}_n, \mathcal{V}_n)$ be a *directed graph* with n nodes $\mathcal{U}_n = \{u_1, \ldots, u_n\}$ and m arcs $\mathcal{V}_n = \{v_1, \ldots, v_m\}$; its *node-arc incidence matrix* $E \equiv E_n = E(\mathcal{G}_n)$ is the $n \times m$ matrix such that $E_{ij} = 1$ if v_j emanates from u_i, $E_{ij} = -1$ if v_j terminates at u_i and $E_{ij} = 0$ otherwise. Hence E has exactly two non-zero elements (a 1 and a -1) in every column.

Given a directed graph $\mathcal{G}$, the linear Minimum Cost Flow (MCF) problem [4] is the Linear Program (LP)

$$\min \left\{ \mathbf{c}^T\mathbf{x} \ : \ E\mathbf{x} = \mathbf{d} \ , \ \mathbf{0} \le \mathbf{x} \le \mathbf{u} \right\} \tag{1}$$

where E is the node-arc incidence matrix of $\mathcal{G}$, $\mathbf{c}$ is the vector of arc costs, $\mathbf{u}$ is the vector of arc upper capacities, $\mathbf{d}$ is the vector of node deficits and $\mathbf{x}$ is the vector of flows. The *flow conservation constraints* $E\mathbf{x} = \mathbf{d}$ express the fact that the flow has to travel in the graph from *sources* (nodes with $d_i > 0$) to *destinations* (nodes with $d_i < 0$), while for the remaining *transhipment nodes* (with $d_i = 0$) the total inbound flow must equal the total outbound flow. This problem has a huge set of applications, either in itself or, more often, as a submodel of more complex and demanding problems [1]. Without loss of generality we can restrict our analysis to connected graphs, as the general case of more than one connected component can be traced back to this case (basically there is a separate LP for each one).

We study graph matrices coming from the application of Interior Point (IP) methods, which have grown a well-established reputation as efficient algorithms for large-scale problems. In these methods, at each step we have to solve linear systems of the form

$$E\Theta E^T\mathbf{x} = \mathbf{b} \ , \tag{2}$$

where E is fixed, while $\mathbf{b} \in \mathbb{R}^n$ and the $m \times m$ diagonal positive definite matrix Θ depend on the IP iteration; as we will not consider (possible) strategies for re-use of information between two different IP iterations, we will disregard this dependence, thus focusing our attention to the solution of (2) at any one fixed iteration. Since each diagonal element of Θ is associated to a specific arc of $\mathcal{G}$, we can consider $\mathcal{G}$ as a *weighted* graph, with Θ specifying the arc weights. In the following we will often use the shorthand $L = E\Theta E^T$, also disregarding the fact that L actually depends on Θ. As the following remark shows, L is closely tied to other well-known graph matrices.

Remark 1. Let $\mathcal{G}' = (\mathcal{U}, \mathcal{V}')$ be the *undirected* graph obtained from $\mathcal{G}$ by ignoring the orientation of the arcs. $\mathcal{G}'$ also is a weighted graph, the weight w_{uv} of each edge $\{u, v\} \in \mathcal{V}'$ being the sum of the weights of all arcs of $\mathcal{G}$ which "collapse" in $\{u, v\}$ (note that multiple parallel arcs are allowed in (1), as they can have different cost and capacity). Then let $A(\mathcal{G}')$ be the symmetric (weighted) adjacency matrix of $\mathcal{G}'$, such that A_{uv} is the weight of the edge $\{u, v\}$ if it belongs to $\mathcal{V}'$ and 0 otherwise. Further let $D(\mathcal{G}')$ be the $n \times n$ the diagonal matrix with $D_{uu} = \sum_{v \in \mathcal{U}} A_{uv}$ (d_u is the node degree of u in the unweighted case). The *Laplacian* of $\mathcal{G}'$ is

$$L(\mathcal{G}') = D(\mathcal{G}') - A(\mathcal{G}') \ ,$$

and it is easy to show that

$$L(\mathcal{G}') = E(\mathcal{G})\Theta(\mathcal{G})E(\mathcal{G})^T \ .$$

Hence, a relevant topological information regarding the original directed graph $\mathcal{G}$ is contained in $E(\mathcal{G})$ as well as in $L(\mathcal{G}')$. The Laplacian of a graph has very many applications in such diverse fields as graph theory, statistics and combinatorial optimization [17, 41, 48].

In most general-purpose LP solvers, the linear systems (2) are solved by means of direct methods, typically the Cholesky decomposition preceded by a heuristic reordering of the columns of E aimed at minimizing the fill-in. For very large, sparse networks this approach is inefficient, as disastrous fill-in (e.g. [13]) may occur which renders the iteration cost unbearable. One thus has to revert to iterative methods instead, but these

approaches can be competitive only if the rate of convergence is sufficiently high. This motivates studies of the extreme singular values of E and of the spectral behaviour of L, since the convergence rate of iterative methods largely depends on the conditioning $\mu(\cdot)$ and more generally on the spectral structure of the matrix (see [35] for an analytic study of such topics). In the first IP iterations, the matrix Θ is close to the identity and the spectral difficulties are mild: $\mu(L) \leq cn^2$, with c absolute constant and the bound attained, up to lower order terms, in the case of linear graphs [35]. However in the last IP iterations the matrix Θ becomes highly unbalanced and the conditioning of L is essentially described by the wild conditioning of Θ. This phenomenon is analyzed in [51] for the case of linear graphs and this analysis is particularly relevant for the numerical solution of (2) through a preconditioned conjugate gradient (PCG) method. Most PCG-based IP algorithms employ *support-graph preconditioners* [62, 6, 8, 9] of the form

$$L_{\mathcal{S}} = E_{\mathcal{S}} \Theta_{\mathcal{S}} E_{\mathcal{S}}^T \ ,$$

where $E_{\mathcal{S}}$ and $\Theta_{\mathcal{S}}$ denote the restriction of E and Θ respectively, on the arcs of a "simple" subgraph $\mathcal{S}$ of $\mathcal{G}$. Basically there are two possible choices for $\mathcal{S}$. The first aims at minimizing the computational burden of inversion (factorization) of $L_{\mathcal{S}}$, which is obtained by choosing it as a spanning tree or as some other chordal-type graph [49, 32, 33]: in this way, the corresponding node-arc incidence matrix is in triangular or block triangular form and $L_{\mathcal{S}}$ can be factorized without fill-in. From a computational viewpoint, spanning trees are the most effective choice in all but the most difficult cases, due to the fact that $\mathcal{S}$ can be chosen as an (approximate) Minimum Spanning Tree in $O(m)$, e.g. by employing the Prim algorithm. The other choice instead aims at selecting $\mathcal{S}$ in such a way that the best possible provable improvement of the preconditioned system conditioning is obtained. This usually reduces to appropriately (recursively) partitioning the graph into "weakly interacting" clusters [59, 44]. Alternatively (or in addition) Steiner preconditioners [37, 43] allow to simplify the task of choosing the right subgraph at the cost of introducing "fake" nodes in the graph ("do not care equations") and therefore solving a slightly larger problem.

While theoretically sound and supported by a sophisticated analysis, the status of "complex" support-graph preconditioners from the computational standpoint is not yet very clear, as some of the hidden constants in the complexity results may be large. This is even more relevant when considering actual implementations of the corresponding approaches, which are complex and not widely available, while "simple" PCG approaches using tree or tree-like subgraphs can be implemented with minor modifications of standard solution methods. Yet, while these preconditioners often work quite well, there are some cases where the convergence rate is slow. Along the lines of [45, 23, 24], the objective of this paper is therefore to evaluate whether "simple" tree-based PCG approaches can be complemented with ideas from the algebraic multigrid (AMG) field [55, 60] —and in particular from multi-iterative techniques [57]—to yield methods that combine a relatively simple implementation and robustness, in the sense of uniformly delivering good performances, without the need of a complex parameter tuning.

2.2. Multi-iterative idea and MG methods

Let $L = L^T \in \mathbb{R}^{n \times n}$ be a positive definite matrix, $\mathbf{b} \in \mathbb{R}^n$ be the right-hand-side and, $l \in (0, n)$ (most often $l \approx \log n$) be the *number of levels*. Fix integers $n_0 = n > n_1 > n_2 > \ldots > n_l > 0$, take $R_{i+1}^i \in \mathbb{R}^{n_{i+1} \times n_i}$ full-rank matrices and consider a class $\mathcal{S}_i$ of iterative methods for n_i-dimensional linear systems. The related *V-cycle method* [61] produces the sequence $\{\mathbf{x}^{(k)}\}_{k \in \mathbb{N}}$ according to the rule $\mathbf{x}^{(k+1)} = \mathcal{MGM}(0, \mathbf{x}^{(k)}, \mathbf{b})$, with $\mathcal{MGM}$ recursively defined as follows:

$$\mathbf{x}_i^{(\text{out})} := \mathcal{MGM}(i, \mathbf{x}_i^{(\text{in})}, b_i)$$

`if(` $i = l$ `) then` `Solve`$(L_l \mathbf{x}_l^{(\text{out})} = \mathbf{b}_l)$

`else`

1 $\mathbf{r}_i := L_i x_i^{(\text{in})} - \mathbf{b}_i$

2 $\mathbf{b}_{i+1} := R_{i+1}^i \mathbf{r}_i$

3 $L_{i+1} := R_{i+1}^i L_i (R_{i+1}^i)^T$

4 $\mathbf{y}_{i+1} := \mathcal{MGM}(i+1, 0_{n_{i+1}}, \mathbf{b}_{i+1})$

5 $\mathbf{x}_i^{(\text{int})} := \mathbf{x}_i^{(\text{in})} - (R_{i+1}^i)^T \mathbf{y}_{i+1}$

6 $\mathbf{x}_i^{(\text{out})} := \mathcal{S}_i^{\nu}\left(\mathbf{x}_i^{(\text{int})}\right)$

Step 1 calculates the residual of the proposed solution. Steps 2, 3, 4, 5 define the *recursive coarse grid correction* by projection (step 2) of the residual, sub-grid correction (steps 3, 4) and interpolation (step 5), while step 6 performs some (ν) iterations of a "post-smoother".

By using the MGM as an iterative technique [55], at the k-th iteration, we obtain the linear systems $L_i\mathbf{x}_i^{(k)} = \mathbf{b}_i^{(k)}$, $i = 0, \ldots, l$, where the matrices $L_i = L_i^T \in \mathbb{R}^{n_i \times n_i}$ are all positive definite. Only the last one is solved exactly, while all the others are recursively managed by reduction to low-level system and smoothing. The procedures $\mathcal{S}_i$ are most often standard stationary iterative methods [64], such as Richardson, (damped) Jacobi, Gauss-Seidel etc., with prescribed iteration matrix $S_i \in \mathbb{R}^{n_i \times n_i}$, i.e.

$$\mathcal{S}_i(\mathbf{x}_i^{(\text{int})}) = S_i\mathbf{x}_i^{(\text{int})} + (I_{n_i} - S_i)L_i^{-1}\mathbf{b}_i^{(k)} \quad , \quad \mathbf{x}_i \in \mathbb{R}^{n_i} \quad , \quad i = 0, \ldots, l-1 \ .$$

If we recursively define the multigrid iteration matrix of level $i = l-1, \ldots, 0$ as

$$\begin{cases} MGM_l = 0_{n_l \times n_l} \\ MGM_i = S_i^{\nu_i}\Big[I_{n_i} - \big(R_{i+1}^i\big)^T\big(I_{n_{i+1}} - MGM_{i+1}\big)L_{i+1}^{-1}R_{i+1}^iL_i\Big] \end{cases} , \tag{3}$$

then $\mathbf{x}_i^{(\text{out})} = MGM_i\mathbf{x}_i^{(\text{in})} + (I_{n_i} - MGM_i)L_i^{-1}\mathbf{b}_i$, so in the finer grid we have

$$\mathbf{x}^{(k+1)} = MGM_0\mathbf{x}^{(k)} + (I_{n_0} - MGM_0)L_0^{-1}\mathbf{b}, \qquad \mathbf{x}^{(r)} = \mathbf{x}_0^{(r)} \ \ \forall r,$$

and MGM_i depends on i but not on any of the $\mathbf{x}_i^{(k)}$ and $\mathbf{b}_i^{(k)}$. For each $i = 0, \ldots, l-1$, the algorithm has essentially two degrees of indetermination:

- the choice of the projectors R_{i+1}^i;
- the choice of the smoothers $\mathcal{S}_i$.

The former, as well as the calculation of the L_i matrices, are performed before the beginning of the V-cycle procedure (pre-computing phase). While this is not necessary in the general multi-iterative approach, we will only consider convergent smoother iterations. Moreover, if the smoother is convergent in the L-norm ($\|\mathbf{x}\|_L^2 = \mathbf{x}^TL\mathbf{x}$, L symmetric and positive definite), then the multigrid iteration matrix has L-norm smaller than that of the smoother. In other words, the multigrid iteration is never worse than the smoother alone [57, 38].

Two further modifications can also be applied:

- using a *pre-smoother*, i.e., adding a step 0 similar to step 6 where a further stationary iterative method is employed;
- allowing, in steps 6 (and 0), the number of smoothing iterations to depend on the level i.

This corresponds to the matrix in (3) being multiplied on the right by a further iteration matrix, i.e.

$$MGM_i = S_{i,\text{post}}^{\nu_{i,\text{post}}}\Big[I_{n_i} - \big(R_{i+1}^i\big)^T\big(I_{n_{i+1}} - MGM_{i+1}\big)L_{i+1}^{-1}R_{i+1}^iL_i\Big]S_{i,\text{pre}}^{\nu_{i,\text{pre}}} \ ,$$

where the number of smoothing steps $\nu_{i,\text{pre}}$ and $\nu_{i,\text{post}}$ depend on the level i. In practice (cf. [2, 58] and the references therein) the application of the pre-smoother accelerates the global convergence substantially, but the explanation of this phenomenon falls outside the convergence theory of the algebraic multigrid and indeed pertains to multi-iterative methods [57]. For instance, looking just at the two-grid method, in the case of the d-dimensional discrete Laplacian, it is easy to prove that the post-smoothing given by the Richardson iteration with $\omega = \omega_1 \equiv 1/4d$ is strongly converging in the subspace of the high frequencies and that the coarse grid correction strongly reduces the error in the low frequencies subspace. Therefore, the combination of the two complementary iterations, which separately are slowly convergent on the global space $\mathbb{R}^n$, leads to a fast convergent two-grid method. However a finer analysis tells us that the global error is now essentially localized in the middle frequencies: the iteration again given by Richardson, but with $\omega = \omega_2 \equiv 1/2d$ is not a smoother: however such a simple method is fast convergent just in the middle frequencies subspace. Therefore its further use in step 6 (or equivalently in step 0) increases very much the "spectral complementarity", so that we obtain a real multi-iterative method, whose global spectral radius is really small. We call an iteration having a spectral behaviour complementary to both the coarse grid correction and the smoother an "intermediate iteration"; an example is the Richardson iteration with $\omega_2 = 2\omega_1$ in the case of the discrete Laplacian. Regarding the

level-dependent number of smoothing iterations [58], it can be easily shown that a polynomial growth with i does not affect the global cost, that remains linear for banded or sparse structures, only changing the constants involved in the big O. This strategy, together with sophisticated preconditioners in the smoothing phases, was the key for developing very effective multigrid solvers for very ill-conditioned IgA Galerkin B-spline matrices [28] and for Sinc-Galerkin matrices [50], where, like in the case of weighted Laplacians, there is the presence of a positive diagonal matrix whose conditioning is extremely high (in fact exponential as the size of the matrix in the Sinc-Galerkin setting). However, while in the standard differential setting there is a gain in using an increasing $\nu_i = \nu_{i,\text{pre}} + \nu_{i,\text{post}}$, for problems with a smaller ill-conditioning (regularized Laplacian in [29]) the method, that achieves the smallest theoretical cost and that minimizes the actual CPU times for reaching the solution with a preassigned accuracy, is the simplest V-cycle with only one step of post-smoothing given by a classical damped Jacobi.

Since several possible choices exist in the implementation of the MG approach, in the remainder of this section we will briefly discuss some initial computational tests, performed in the context of MCF problems, that provide a guidance on how to choose at least the two main components: projectors and smoothers.

2.3. Ghost node

Let $\mathbf{e} = \mathbf{e}_{(n)}$ be the all-ones vector of length n. It is immediate to realize that $\mathbf{e}^T E = \mathbf{0}^T$ and therefore $\mathbf{e}^T L = 0^T$, i.e. $\text{rank}(L) = n - 1$. However $\mathbf{e}^T \mathbf{d} = 0$ as well (for otherwise (1) cannot have any feasible solution [1]), and this property is transmitted to the right-hand sides $\mathbf{b}$. Hence by the Rouché-Capelli Theorem the linear system (2) has ∞^1 solutions of the form $\mathbf{x}(\alpha) = \hat{\mathbf{x}} + \alpha\mathbf{e}$ for $\alpha \in \mathbb{R}$. Thus, let E' and $\mathbf{b}'$ be obtained by E and $\mathbf{b}$ respectively by erasing any one row and L' obtained by E' as usual. A solution to (2) can be obtained by solving the *cut system*

$$L'\mathbf{x}' = \mathbf{b}' \tag{4}$$

and setting $\hat{\mathbf{x}} = (\mathbf{x}', 0)$. Intuitively this amounts at creating a "ghost node", the one corresponding to the deleted row, in the graph, as the columns of E corresponding to arcs entering (leaving) the ghost node will only have a 1 (-1) without the corresponding -1 (1).

It is well-known that, since L is a semidefinite positive matrix, CG and PCG approaches (started from the zero vector) solve (2) in the least-squares sense, i.e. find

$$\bar{\mathbf{x}} = \underset{\mathbf{x}\in\mathbb{R}^n}{\text{argmin}} \|L\mathbf{x} - \mathbf{b}\|_2 \qquad \text{or equivalently} \qquad \|\bar{\mathbf{x}}\|_2 = \min_{\alpha\in\mathbb{R}} \|\hat{\mathbf{x}} + \alpha\mathbf{e}\|_2 \ . \tag{5}$$

This is, in general, not true for other methods. Hence using CG or PCG allows to work on the original graph at all levels but the last one, where eliminating one row (i.e. passing from (2) to (4)) to recover L'_l is necessary since a non-singular matrix is required by direct methods. Thus [P]CG can maintain our graph "sheltered from ghosts" which, as we will see, can have a positive impact on performances.

2.4. Smoothers

In accordance with the most successful strategies in the differential case, we tried as smoothers a combination of classical iterations. We recall that the discrete Laplacian can be viewed as a special instance of a graph matrix. In addition, by the analysis in [35], all the (unweighted) graph matrices with $\Theta = I$ share the property that the small eigenvalues are related to smooth eigenvectors. In other words, as in the differential setting, the degenerating subspace is located essentially in the low frequencies. Thus, following multi-iterative idea [57], we performed an extensive search for a combination of "simple" pre- and post-smoothers (e.g., relaxed Jacobi or relaxed Gauss-Seidel with different choices of the relaxation parameter) which was effective from the point of view of spectral complementarity. However, we were not able to find it. All combinations of classical pre- and post-smoothers we tested were clearly ineffective, especially in the last iterations of the IP process where the Θ becomes very unbalanced, with very large entries ($\approx$ `1e+6`) and very small entries ($\approx$ `1e-10`).

A similar occurrence was already experienced in [50], where the distribution of the nodes in the Sinc-Galerkin method induced very unbalanced diagonal entries. In that application, the only successful strategy was the combination of PCG smoothers with sophisticated and powerful preconditioners, in connection with the classical projection used in the differential context. As will be discussed in §3.3, that previous experience is strongly related with the current setting by the fact that the full weighting operator is a special case of the ones studied here. This means that even if the projection operator is appropriate, unlike the simple ones we used in our initial tests (cf. §2.5 and [26]), "simple" smoothers may not be enough to obtain an efficient

approach when the system is ill-conditioned. This was in fact later confirmed, as reported in §4.4, when testing the solver of [45] on our problems. In that code, a very sophisticated and principled choice of the projection operator is performed, but it is used in conjunction with standard smoothers; the corresponding results were not competitive. All these experiences justify our choice of focusing our attention on more powerful smoothers, that is, the PCG family with a large set of specialized preconditioners:

- diagonal;
- incomplete Cholesky factorization with tolerance parameter droptol (non-zero fill-in);
- tree-based preconditioners.

However, as it will be clear from the next subsection, the problem with general graph matrices is more complicated than the one in [50], since the graph structure is not nearly as regular as those arising in the context of differential operators.

Regarding the computational burden, we assume that the cost at every level of the MG iteration is linear with respect to the dimension (let us say bounded by $O(n_i) = \gamma n_i$) plus the recursion part and we assume that the size is halved at every recursion step. Concerning the number of smoothing steps, we consider the growth function $\nu_{i,\text{pre}} = i^k$ that is we make more smoothing steps at the inner levels and we assume that the cost of the smoothing iteration is responsible of the linear cost $O(n_i) = \gamma n_i$. The total work W_i at level i required to perform a MG iteration is then given by the recursive law

$$W_i \leq \gamma n_i (l + 1 - i)^k + W_{i-1}, \quad n_{i-1} = \frac{n_i}{2},$$

with W_0 constant. Therefore, setting $n = n_l$, at the finest level $l \approx \log(n)$ we have

$$W_l \leq \sum_{i=1}^{l} \gamma \frac{n}{2^{i-1}} i^k + W_0 < \gamma n \sum_{i=1}^{\infty} \frac{i^k}{2^{i-1}} + W_0 \ . \tag{6}$$

The rightmost series in (6) is convergent; its initial values for $k = 0, 1, 2, 3, 4, 5$ are respectively 2, 4, 12, 52, 300 and 2164. This estimate has to be multiplied by 2 when employing both pre- and post-smoother, as in our case. Numerical experiments with different values of k ($\in \{0, 1, 2\}$) clearly showed that $k = 0$ results in the best performances [26], which was therefore the setting used in all the subsequent test.

2.5. Projectors

Our initial interest was to verify whether standard operators with good performances in the context of differential equations, such as the very classical Full Weighting Operator (FWO)

$$R_{\text{FWO}} = \frac{1}{4} \begin{pmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & 2 & 1 \end{pmatrix}, \tag{7}$$

would perform equally well in the context of general graph matrices. The preliminary results quickly made it clear that this was not the case [26]: in fact, the corresponding MGM is not efficient even for these easy graphs and $\Theta = I$, which is generally the easiest case because it shows a moderate conditioning [35]. The results are also confirmed by the fact that using more sophisticated choices, such as quadratic approximation [27] with stencil

$$\frac{1}{16}[1, 4, 6, 4, 1]$$

and even cubic interpolation [61] with stencil

$$\frac{1}{32}[-1, 0, 9, 16, 9, 0, -1]$$

does not lead to better results (actually even to worse ones). Thus the differential setting is of no help in our graph problem. This is due to the fact that the matrix at lower levels is far away from a graph matrix and this destroys the structure which the MGM relies upon.

All this pushed us towards the development of the different operators described in this paper. In particular we quickly focused our attention on Aggregation Operators (AGO) of the generic form

$$R_{\rm AGO} = \begin{pmatrix} 1 & 1 & 1 & & & & & \\ & & & 1 & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & 1 & 1 & 1 \end{pmatrix}, \tag{8}$$

which *preserve the graph structure* in the recursion, i.e.

$$R_{\rm AGO} E \Theta E^T R_{\rm AGO}^T \;=\; R_{\rm AGO} L(\mathcal{G}) R_{\rm AGO}^T \;=\; L(\mathcal{G}')$$

is the Laplacian of a new graph $\mathcal{G}'$ corresponding to the *aggregation of nodes* of the original graph (see Definition 2). Similar operators have been used with good results in different contexts, for example when designing multigrid solvers for Markov Chains [20]. Numerical testing clearly showed that MG approach with a simple structure-preserving projector appeared to be competitive. Furthermore the tests showed that, for general graphs and arc weights (such as these of MCF matrices at final IP iterations), the only effective smoothers are preconditioned Krylov methods. Therefore the only ingredient that may possibly make the V-cycle competitive is the right choice of the projectors. Since most of the effective preconditioners for this class of systems are of support-graph type, it is interesting to investigate the class of projection operators, preserving the graph structure — and therefore using support-graph preconditioners — at all levels of the MG approach. This is precisely what the next section is devoted to.

3. Graph operators theory

In this section we study the conditions under which projection operators preserve the graph structure of the matrix. We start with some preliminary definitions.

Definition 2. An operator R is called a *graph* operator if, given any incidence matrix E,

$$RE = E'\Theta' \;, \tag{9}$$

where E' is an incidence matrix and Θ' is an invertible diagonal matrix.

According to (2), one has

$$RE\Theta E^T R^T \;=\; E'\Theta'\Theta\Theta' E'^T \;=\; E'\tilde{\Theta}E' \;,$$

where $\tilde{\Theta} = \Theta'\Theta\Theta'$ is a diagonal matrix with positive diagonal entries, if Θ is (we do not need the diagonal entries of Θ' to be positive). Consequently it is evident that if the restriction R_{i+1}^i in the multigrid method is a graph operator, then it preserves the graph structure at the lower levels, in the sense that the projected problem is still a (smaller) weighted Laplacian, thus allowing an efficient recursive strategy in the MG solver. A weaker notion is:

Definition 3. An operator R is called a *graph operator for a given matrix* E if (9) holds for E, although it may not hold for all possible incidence matrices.

Definition 4. An operator R is called *admissible* if it does not have any column with all zero elements and any row with all equal elements.

The rationale for this definition is that an all-zero column means that a node is "ignored", so that the projection ($\mathbf{r}_i = (R_{i+1}^i)^T \mathbf{r}_{i+1}$) cannot produce any correction of the error. Symmetrically, a row with all equal elements in R means that RE has an all-zero row, i.e. E' in (9) has an isolated node. Furthermore note that AMG conditions impose that R has to be of full rank, so that in any case at most only a unique row could have all equal elements. We remark that Definition 4 implies that $\mathrm{nz}(R) \geq n$, where $\mathrm{nz}(\cdot)$ denotes the number of non-zero elements. This leads to the following refinement of the concept.

Definition 5. An admissible graph operator R such that $\mathrm{nz}(R) = n$ is called *minimum*.

Of course, any admissible operator with $\mathrm{nz}(R) > n$ is *non-minimum*, and the set of graph operators is partitioned between the two subsets. Similar definition can be given for graph operators w.r.t. a specific matrix E.

3.1. Necessary and sufficient conditions for graph operators

We now are interested in a characterization of the set of graph operators. In this section we will employ the following notation: $A^{[k]}$ is the k-th column of a generic matrix A, E is the $n \times m$ node-arc incidence matrix of the underlying *connected* graph $\mathcal{G}$ (cf. §2.1), for each $k = 1, \ldots, m$ the tail and head nodes of the corresponding arc (the row indices corresponding to 1 and -1 in $E^{[k]}$) are i_k and j_k respectively, R is a $n' \times n$ operator with $n' < n$ (usually $n' \approx n/2$) so that $F = RE$ is a $n' \times m$ matrix, $e' = e_{(n')}$ is the all-ones vector of length n'.

Definition 6. R is a *constant column sums* (CCS) operator if the sum of elements of every column is constant, i.e. there exists $\rho \in \mathbb{R}$ such that $(\mathbf{e}')^T R = \rho \mathbf{e}^T$.

Definition 7. R is a *zero column sum* (ZCS) operator for E if $(\mathbf{e}')^T F = (\mathbf{e}')^T RE = \mathbf{0}^T$ (i.e R conserves the property of E that all columns have zero sum). R is a *ZCS operator* if it is a ZCS operator for all graph matrices E.

Theorem 1. *Let E be an incidence matrix. R is a ZCS operator for E if and only if R is a CCS operator.*

Proof. [$\Rightarrow$] We want to prove that $\mathbf{r}^T = (\mathbf{e}')^T R = \rho \mathbf{e}^T$ for some $\rho \in \mathbb{R}$. Suppose otherwise and let i, j be two indices such that $r_i \neq r_j$. Because $\mathcal{G}$ is connected, there exists at least one path in $\mathcal{G}$ having the nodes i and j as endpoints. It cannot be that $r_p = r_q$ for every arc $k = (p, q)$ belonging to the path, as this would imply $r_i = r_j$. Hence there must be $k = (p, q)$ such that $r_p \neq r_q$, which leads to

$$0 = (\mathbf{e}')^T F^{[k]} = (\mathbf{e}')^T (RE)^{[k]} = \mathbf{r}^T E^{[k]} = r_p - r_q \neq 0$$

that is a contradiction.

[$\Leftarrow$] For every CCS operator R and for every incidence matrix E we have $(\mathbf{e}')^T F = (\mathbf{e}')^T RE = \rho \mathbf{e}^T E = \mathbf{0}^T$. □

Corollary 1. *If R is a ZCS operator for E then it is a ZCS operator.*

Proof. Since R is a ZCS operator for E, by Theorem 1 R is also a CCS operator. Now, by the second point of the previous proof, it follows that R is a ZCS operator. □

Remark 2. Corollary 1 implies that the two forms in Definition 7 are equivalent.

Corollary 2. *Let E be an incidence matrix. Any graph operator for E is a CCS operator.*

Proof. Since R is a graph operator for E, we have (cf. §2.3)

$$(\mathbf{e}')^T F \;=\; (\mathbf{e}')^T RE \;=\; (\mathbf{e}')^T E' \Theta'^T \quad ,$$

i.e. R is a ZCS operator for E. Hence by Theorem 1 it is a CCS operator. □

Corollary 3. *Let R be a minimum operator. Then, up to a scalar factor ρ, R is a binary matrix, i.e. $R \in \{0, 1\}^{n' \times n}$.*

Proof. By Definition 5 every column of the graph operator R only has one non-zero element and by Corollary 2 $(\mathbf{e}')^T R = \rho \mathbf{e}^T$, hence every non-zero entry must be equal to ρ. □

Definition 8. R is a *difference operator* for a set $\mathcal{I} = \{(i_1, j_1), (i_2, j_2), \ldots\}$ of pairs of node indices if for every $(i_k, j_k) \in \mathcal{I}$ the vector $R^{[i_k]} - R^{[j_k]}$ has either 0 or 2 non-zero elements.

Lemma 1. *For each $k = 1, \ldots, m$, $F^{[k]} = (RE)^{[k]} = R^{[i_k]} - R^{[j_k]}$.*

Proof. Immediate by the definition of E. □

Theorem 2. *Let R be a CCS operator. R is a graph operator for E if and only if R is a difference operator for the set $\mathcal{I} = \mathcal{V}$ of the arcs of $\mathcal{G}$.*

Proof. [⇒] Assume by contradiction that R is not a difference operator for $\mathcal{I} = \mathcal{V}$. There exists an arc $k = (i_k, j_k) \in \mathcal{V}$ such that $R^{[i_k]} - R^{[j_k]}$ has neither 0 nor to 2 non-zero elements. This by Lemma 1 implies that the same holds for $F^{[k]} = (RE)^{[k]}$, therefore R is not a graph operator, since (9) cannot hold.
[⇐] If R is a difference operator for $\mathcal{V}$, then for each $k = (i_k, j_k) \in \mathcal{V}$ the column $F^{[k]} = R^{[i_k]} - R^{[j_k]}$ has either 0 or 2 non-zeroes. Assume the latter and let p and q be the row indices of the non-zeroes in $F^{[k]}$. Since R is a CCS operator, by Theorem 1 we have

$$0 = (\mathbf{e}')^T F^{[k]} = F_{pk} + F_{qk} \Rightarrow F_{pk} = -F_{qk} \ .$$

We can then scale $F^{[k]}$ by using the k-th diagonal entry of Θ', finally yielding $F = E'\Theta'$ where E' is an incidence matrix (possibly with "empty arcs"). Therefore R is a graph operator. □

Remark 3. It is easy to verify that minimum operators are CCS operators and difference operators for any set $\mathcal{I}$.

3.2. Admissible graph operator theorem

In the following we prove a somewhat surprising result concerning admissible graph operators.

Definition 9. $C \in \mathbb{R}^{n' \times n}$ is called a *row matrix* if each of its rows is a scalar multiple of $\mathbf{e}^T$, i.e. $C = c\mathbf{e}^T$ for some $c \in \mathbb{R}^{n'}$.

Lemma 2. *Let $A \in \mathbb{R}^{n' \times n}$, $\mathbf{s} \in \mathbb{R}^{n'}$ and $Q \in \mathbb{R}^{n' \times n}$ defined as*

$$Q_{ij} \ = \ \begin{cases} 0 & \text{if } A_{ij} = s_i \\ 1 & \text{otherwise} \end{cases} \ .$$

Then for any pair (i, j) the number of non-zero elements of $A^{[i]} - A^{[j]}$ is equal to the number of non-zero elements of $Q^{[i]} - Q^{[j]}$ plus the number of indices k such that $A_{ki} \neq A_{kj}$ and $Q_{ki} = Q_{kj} = 1$.

Proof. For any k, denote $\alpha_k = A_{ki} - s_k$ and $\beta_k = A_{kj} - s_k$. If $Q_{ki} \neq Q_{kj}$, it means that $Q_{ki} = 1$ and $Q_{kj} = 0$ or vice-versa; hence $\alpha_k \neq 0$ and $\beta_k = 0$ or vice versa. Therefore $A_{ki} - A_{kj} = (A_{ki} - s_k) - (A_{kj} - s_k) = \alpha_k - \beta_k \neq 0$. When $Q_{ki} = Q_{kj} = 0$, one has $\alpha_k = \beta_k = 0$, and therefore $A_{ki} - A_{kj} = 0$. In the only remaining case $Q_{ki} = Q_{kj} = 1$ one directly counts whether $A_{ki} \neq A_{kj}$ or not, hence the proof is finished. □

Theorem 3. *If R is an admissible graph operator and $n' > 2$ then*

$$R = M + C \ ,$$

where C is a row matrix and M is a minimum operator.

Proof. Since R is admissible, after a proper reordering of rows and columns of R, we have $R = [R_1 \mid R_2]$ where R_1 is a $n' \times n'$ matrix such that $R_{ii} \neq 0$ for all i. For minimum operators, without loss of generality, $R_1 = \rho \cdot I$, where ρ is a scalar factor and R_2 contains only copies of some columns of R_1.

We arbitrarily select one column of R — say $R^{[t]}$ — as s and we apply Lemma 2 to R. Since R is admissible, there cannot be an all-zeroes row in Q, because R does not have any row with all equal elements. Since R is also a graph operator, (9) holds for *any graph* $\mathcal{G}$. By Theorem 2 R is therefore a difference operator for *any set* $\mathcal{I}$, hence in particular for $\mathcal{I} = \{(1, t), (2, t), \ldots, (n, t)\}$ (the star tree rooted at t). By the definition of difference operator, in each column of Q there are either 0 or 2 ones. Moreover, for any pair $Q^{[i]}$ and $Q^{[j]}$ of non-zero columns at least one of the two non-zeroes must be in the same row. In fact, assume by contradiction that $Q^{[i]}$ and $Q^{[j]}$ have all their non-zeroes in different positions: this means that there exists for Q a difference vector $Q^{[i]} - Q^{[j]}$ with 4 non-zero elements and by Lemma 2 the same holds for R, since there is no index k such that $Q_{ki} = Q_{kj} = 1$.

These two facts imply that we can reorder the rows and columns of Q (which corresponds to reordering R) by putting the column t as the first one and all the others appropriately in such a way that $Q = [Q_1 \mid Q_2]$, where Q_1 is the following $n' \times n'$ matrix

$$Q_1 \ = \ \begin{pmatrix} 0 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

and Q_2 contains only copies of some columns of Q_1. Now assume that R has in fact been reordered (if necessary) to match Q, consider the first row in (the reordered) R and pick any two column indices $h \neq j$ such that $Q^{[h]}$ and $Q^{[j]}$ are not all-zero; hence $Q_{1h} = Q_{1j} = 1$ (remind that the columns in Q_2 are copies of these in Q_1, hence all the non-zero ones have a 1 in the first row). We claim that $R_{1h} = R_{1j}$. We start with the case where the other two non-zeroes are in different rows, i.e. $Q_{ph} = 1$ and $Q_{qj} = 1$ for $1 < p \neq q > 1$: by Lemma 2 we have a difference vector for R with 2 or 3 non-zero elements, depending on whether $R_{1h} = R_{1j}$ or not, which proves our claim since R is a difference operator. The case when $p = q$ easily follows by transitivity, considering one further column h (which must exists) that has its other non-zero element in a different row: $R_{1i} = R_{1h}$ and $R_{1j} = R_{1h}$, so $R_{1i} = R_{1j}$. Note that the hypothesis $n' > 2$ is crucial in this part of the proof.

Consider now any row $i > 1$. Needless to say, all entries R_{ij} such that $Q_{ij} = 0$ have the same value (that of R_{ik}). Therefore we know that R has an arrangement as $R = [\, R_1 \mid R_2 \,]$, where

$$R_1 = \begin{pmatrix} r_1 & c_1 & c_1 & \cdots & c_1 \\ c_2 & r_2 & c_2 & \cdots & c_2 \\ c_3 & c_3 & r_3 & \cdots & c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n'} & c_{n'} & c_{n'} & \cdots & r_{n'} \end{pmatrix}$$

and each column $R^{[j]}$ of R_2 has the form $[c_1, c_2, c_3, \ldots, c_{n'}]^T$ except (possibly) for one unique element, $R_{ij} = r_j \neq c_i$, at the unique row $i > 1$ (if any) such that $Q_{ij} = 1$. Consider any two column indices $h \neq j$ such that $Q_{ih} = Q_{ij} = 1$: we claim that, again, $R_{ih} = R_{ij}$, i.e. that all columns of R_2 are copies of some column of R_1 (the one having the non-zeroes in Q in the same position). This comes from the fact that, being a graph operator, R is also a CCS operator: $(\mathbf{e}')^T R^{[j]} = (\mathbf{e}')^T R^{[h]} = \rho$. Then one has

$$\rho = (\mathbf{e}')^T R^{[j]} = \sum_{p \neq i} c_p + r_j \qquad \text{and} \qquad \rho = (\mathbf{e}')^T R^{[h]} = \sum_{p \neq i} c_p + r_h$$

whence $R_{ih} = r_h = \rho - \sum_{p \neq i} c_p = r_j = R_{ij}$. Again from the fact that R is a CCS operator, we have

$$\rho = (\mathbf{e}')^T R^{[1]} = r_1 + c_2 + \sum_{p>2} c_p = (\mathbf{e}')^T R^{[2]} = c_1 + r_2 + \sum_{p>2} c_p$$

and similarly for all pairs $(i, i+1)$ with $i < n'$, whence $r_1 - c_1 = r_2 - c_2 = \ldots = r_{n'} - c_{n'}$. Let us call the common value $r_i - c_i = \alpha$: it must be $\alpha \neq 0$, for otherwise one would have $r_i = c_i$ for all i, i.e. $R = C = c\mathbf{e}^T$, contradicting the hypothesis that R is admissible (*each* row would have all equal elements). Hence $M = R - C \neq 0$. In particular M has exactly one non-zero per column, no all-zero rows, and all its non-zeroes have the same value α. Thus M clearly is a minimum operator (cf. Corollary 3). □

Remark 4. The action on an incidence matrix E of any graph operator R and of the related minimum operator M (see Theorem 3) is the same. In fact, $F = RE = ME + CE = ME$.

Remark 5. The hypothesis $n' > 2$ in Theorem 3 is necessary, as for $n' = 2$ the thesis does not hold. In fact, the conditions which have to be satisfied are

- $R_{1j} + R_{2j} = \rho$ for all j,
- $R_{1j} = R_{1k}$ if and only if $R_{2j} = R_{2k}$ for all j and k,

and these do not prevent from choosing all different R_{ij}, so in general $R \neq M + C$. An easy counterexample with $m = 5$ is for instance

$$R = \begin{pmatrix} 0.1 & 0.8 & -1 & 1 & 0.25 \\ 0.9 & 0.2 & 2 & 0 & 0.75 \end{pmatrix} .$$

Corollary 4. *Let R be an admissible graph operator. If R is a binary matrix, then either $R = M$ or $R = \mathbf{e}'\mathbf{e}^T - M$ where M is a minimum operator.*

Proof. By Theorem 3 $R = M + C$. By Corollary 3 M is a binary matrix, up to a scalar factor α. Hence either $C = 0$ and $R = M$ or C is the matrix with all entries equal to 1 and $R = C - M$ is the complement of the minimum operator M (the one having 0 where M has 1 and vice-versa). □

We finish this section by noting that if the original graph $\mathcal{G}$ is connected (as is the case in our applications), then so is the "restricted" graph $\mathcal{G}'$.

Theorem 4. *Let R be an admissible graph operator for $E(\mathcal{G})$. Then $\mathcal{G}'$ is a connected graph.*

Proof. By Remark 4 $F = RE = ME$, hence R acts on E as a minimum operator; these aggregate nodes, which can be adjacent (cf. Definition 10) or not. Clearly contracting nodes in an already connected graph produces a connected graph: each path in $\mathcal{G}$ corresponds to a (possibly non-simple) path in $\mathcal{G}'$. In other words, connectivity in $\mathcal{G}'$ is the same as connectivity in the graph obtained by adding to $\mathcal{G}$ arcs forming cliques within the aggregated nodes. □

3.3. Interpretation: the FWO operator for Poisson problems

The above results help heuristically in understanding why the full weighting operator is a good choice for Poisson problems: basically it is a graph operator for the corresponding very special graph matrix. Recalling its algebraic expression in (7), omitting the constant 1/4, and leaving out the first and the last column, one immediately notes that the FWO is a CCS operator with constant 2. The fundamental observation now is that the matrix associated with the Poisson problem has the EE^T form with the elimination of the first and the last row and the first and the last column:

$$(L_{\text{Poisson}})_n = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}_{n\times n} = P_n E_n E_n^T P_n^T \ ,$$

where
$$E_n = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & \ddots & & \\ & & \ddots & 1 & \\ & & & -1 & 1 \\ & & & & -1 \end{pmatrix}_{(n+2)\times(n+1)} , P_n = \begin{pmatrix} 0 & | & I_n & | & 0 \end{pmatrix}_{n\times(n+2)} .$$

The incidence matrix E is associated to the linear graph with $n+2$ nodes, in which every node points to the next one, as depicted in Figure 1.

Figure 1: Linear graph with $n+2$ nodes. Nodes colored in grey are related to the Dirichlet boundary conditions.

Indeed, R_{FWO} — which is clearly an admissible operator — is a difference operator for the set $\mathcal{I} = \{(2,3), (3,4), \ldots, (n-2, n-1)\}$, as the difference vector is

$$\begin{pmatrix} 2 \\ \\ \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} \\ 2 \\ \\ \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} .$$

The exceptions represented by the first and the last column, in which the difference vector has one non-zero element, do not contradict our results, as they can be explained by the elimination of rows and columns related to the Dirichlet boundary conditions. By Theorem 2 we conclude that R_{FWO} is a graph operator for the matrix E of Poisson problems (cf. Definition 3), although not a graph operator in general. Moreover, by Theorem 4 also the graph associated to $E' = E_{n'}$ is connected; even better, such graph is still a linear graph. In fact,

$$R_{\text{FWO}}(L_{\text{Poisson}})_n P_{\text{FWO}} = \frac{1}{4}(L_{\text{Poisson}})_{n'} = \frac{1}{4} P_{n'} E_{n'} E_{n'}^T P_{n'}^T$$

where $P_{\mathrm{FWO}} = 2(R_{\mathrm{FWO}})^T$ is the operator of linear interpolation, $E_{n'}$ is the incidence matrix associated to the linear graph with $n'+2$ nodes, and $P_{n'}$ is defined as above. Thus the FWO projection preserves the Poisson structure and so it is not surprising that it does work well in the Poisson case. However for general graphs, the structure of weighted Laplacian is not conserved at all by the FWO projection. Consequently, we are not surorised that the related multigrid performances are not good (see the discussion in Section 2.5). Our heuristic indication is to consider only projections that preserve the weighted Laplacian structure.

4. Projection operators

In this paragraph we will review several proposals for projection operators in the context of Laplacian matrices. For each of them we describe the restriction operator $R \in \{0,1\}^{n' \times n}$, where n' is a fraction of n (n actually is n_k, i.e. the matrix dimension at the k-th level). The value $d = n - n'$ is called *descent parameter* and usually $d \approx n' \approx n/2$.

4.1. Iterative pairwise minimal operators

Many different minimal operators can be constructed by iterating the simplest possible form of aggregation, that is the one given by the *Pair Operator*

$$R(i,j) = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & \cdots & 1 & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix}$$

w.r.t. the pair (i,j); in other words, by selecting a set $\mathcal{R}$ of pairs of nodes (with $|\mathcal{R}| = d$) and defining $R_{\mathcal{R}} = \prod_{(i,j)\in\mathcal{R}} R(i,j)$. For instance one could define *oblivious* operators that are independent of the graph they are applied to, such as the Couple Operator aggregating successive nodes ($\mathcal{R} = \{(1,2),(3,4),\ldots\}$) or the Random Operator where $\mathcal{R}$ is randomly chosen. Oblivious operators are attractive from the computational viewpoint because they can be computed a-priori, therefore they are as cheap as possible. However their effectiveness is strongly influenced by the structure of the underlying graph $\mathcal{G}$: in order to construct "robust" operators, *adaptive* choices must be considered which depend on the values of the matrix at hand (topological structure of the graph and weights of the arcs too), despite the fact that they can be more costly to determine [26]. Therefore a non-trivial trade-off between increase of the performances and computational cost has to be determined. In choosing the rules to determine the elements of $\mathcal{R}$, it is likely a wise choice to preserve as much as possible the topological structure of the graph.

Definition 10. A graph operator R is a *contraction operator* for E if it aggregates exclusively adjacent nodes in $\mathcal{G}$.

Remark 6. R_{FWO} is a contraction operator for the matrix E of Poisson problem.

Contractor operators are intuitively attractive, since they do not "mess up" with the structure of the graph. Indeed we tested a simple but effective scheme for choosing the pairs (i,j) to aggregate:

- select i either at random or as the diagonal entry of L with the least (or the greatest) value;
- select j either at random or as the index of the non-zero off-diagonal entry in the i-th column with the least (or the greatest) *absolute* value (since $L_{ij} = -\Theta_{ij} < 0$).

Note that, when the process is repeated, the (iteratively) aggregated matrix

$$R(i,j)LR(i,j)^T$$

is looked at instead of the original L. This gives rise to nine "$\{x,y\}$" projectors with x and y chosen between "rand", "min" and "max". Our preliminary results [26] showed that $\{x,\max\}$ projectors substantially outperform the corresponding variants, where j is selected either at random or by looking at arcs with small weight.

Thus in our experience using contractor operators turns out to be better then aggregating non-adjacent nodes. Furthermore, as with subgraph-based preconditioners, choosing arcs with large weight for $\mathcal{R}$ appears to be the best option.

It is clear that different, and possibly more sophisticated, choices of R may exist. For instance, every possible pair (i, j) actually defines the 2×2 minor

$$L^{ij} = \begin{bmatrix} L_{ii} & L_{ij} \\ L_{ji} & L_{jj} \end{bmatrix} \quad ,$$

where $L_{ii} > 0$, $L_{jj} > 0$, $L_{ij} = L_{ji} < 0$ and $\det(L^{ij})$ is positive. Because weak dominance for rows holds, that is $\sum_{j \neq i} |L_{ij}| \leq L_{ii}$ with strict inequality at least for one index (if R is a graph operator, then this is true at every MGM level), it follows that $|L_{ij}| \leq \min(L_{ii}, L_{jj})$. The choice of j of $\{x, \max\}$ operators implies that, if we suppose that $\min(L_{ii}, L_{jj}) = L_{jj}$, then $L_{ij} \approx L_{jj}$ and

$$\det(L^{ij}) = L_{ii}L_{jj} - L_{ij}^2 \approx (L_{ii} - L_{jj})L_{jj} \quad ,$$

i.e. if $L_{ii} \approx L_{jj}$ then one aggregates a badly conditioned part, whereas if $L_{ii} \gg L_{jj}$ then one aggregates a nicely conditioned part. Hence one may use quantities related to $\det(L^{ij})$, such as "normalized" versions like $\det(L^{ij})/(L_{ii}^2 + L_{jj}^2)$ or $\det(L^{ij})/(L_{ii}L_{jj})$, which measures how well or badly conditioned the 2×2 minor is, to understand how promising a (i, j) pair is. This gives rise to max-minor or min-minor operators, depending on whether one chooses to preferentially aggregate well-conditioned or ill-conditioned minors.

Our experience with these two variants is that aggregating badly conditioned minors is by far the most effective variant. The choice of the (i, j) pair is done as in the previous case: first i is selected with either one of the three above strategies, then j is selected in $O(n)$ ($O(1)$ for sparse graphs) as the one giving the most ill-conditioned minor. Note that it would be possible to determine the "overall best" minor operator by looking at all arcs in $\mathcal{G}$, but that would have a $O(m)$ cost. The $\{x, \text{min-minor}\}$ operator seems to somewhat improve upon the previous $\{x, \max\}$ ones. In particular its performances seems to be even less depending on the choice of i, so the average behaviour is somewhat better. This is especially true when treating extremely ill-conditioned problems, which therefore makes such a strategy our preferred choice. In the rest of the paper we will refer to these projection techniques respectively as *Max* and *Minor* aggregation.

One may wonder whether non-minimum admissible graph operators (which exist by §3.2) could be preferable to minimum ones. Since they are row matrix "perturbations" of minimum operators, these could e.g. by chosen by selecting the minimum operator first, and the perturbation next. In practice, we did not identify any promising way to construct non-minimum operators. All our attempts with non-minimum operators invariably gave either equivalent or worse results than these with minimum ones [26]. Remark 4 may provide a good part of the rationale for such observed results, since for the most part the effect of a non-minimum operator is similar to that of the corresponding minimum one.

4.2. Strength-based aggregation operators

Different projection operators have been defined for different but related problems. For instance, a series of papers [20, 21, 22, 23, 24] by De Sterck and his collaborators examine *strength-based operators* in the context of the solution of problems of the form

$$B\mathbf{x} = \mathbf{x} \quad , \quad \mathbf{x} \geq \mathbf{0} \quad , \quad ||\mathbf{x}||_1 = 1 \quad ,$$

where B a irriducible Markov matrix, which can be rewritten as $L\mathbf{x} = 0$, where $L = I - B$. The techniques proposed in these papers are all based on the concept of *strong dependence*; namely, node i strongly depends on node j if

$$-L_{ij} \geq \theta \left[\max_{k \neq i} -L_{ik} \right]$$

where $\theta \in [0, 1]$ is *strength threshold parameter*. This says that the coefficient L_{ij} is comparable in magnitude to the largest off-diagonal coefficient in the i-th equation; in other context, the same concept is described "in reverse" by saying that the node j *strongly influences* the node i. Let us immediately remark that this notion is strongly influenced by the choice of the parameter θ which is far from being trivial, as discussed in the computational section. Taking into consideration the latter, different aggregation strategies can be developed based on this notion.

- *Strength aggregation.* The algorithm, presented in [20], is based on the notion that "important states", i.e. nodes i that have a large value in the current iterate $\mathbf{x}$, are good candidate "seed points" for new aggregates and that states that are strongly influenced by the seed point of an aggregate are good candidates to join that aggregate. The idea amounts then in choosing, among the unassigned nodes, the one which has the largest value in the current iterate $\mathbf{x}$; a new aggregate is formed, and all unassigned nodes that are strongly influenced by that seed point are added to the aggregate. This process is repeated until all nodes are assigned to one aggregate. In the original implementation, the algorithm recompute aggregates at every level of every V-cycle. In our experience this did not lead to good results, while a similar approach using $\mathbf{b}$ instead of $\mathbf{x}$ (and therefore run only once before applying MGM) proved to be competitive.

- *Neighborhood aggregation.* This algorithm, proposed in [63] and used in [23], is based on the undirected version of strong influence: the (undirected) pair of nodes $\{i, j\}$ is considered to be *strongly connected* if at least one of the nodes is strongly dependent on the other. This gives rise to the *strong neighborhood* $\mathcal{N}_i$ of node i — the set of all points that are strongly connected to i — and to the two-phase neighborhood-based aggregation algorithm. In the first phase, the algorithm assigns entire neighborhoods to aggregates: for each node i, if none of the nodes of $\mathcal{N}_i$ have been assigned yet, then a new aggregate is formed and all the nodes in $\mathcal{N}_i$ are assigned to the aggregate. This phase terminates with at least one aggregate formed, and possibly some non-assigned nodes: then in the second phase each of the remaining nodes is assigned to the aggregate it is "most connected" to, i.e. the one having the largest number of nodes in $\mathcal{N}_i$.

- *(Double) Pairwise aggregation.* This pairwise aggregation algorithm, proposed in [52] and used in [24], is based on the "directed version" of strong neighborhoods, i.e. on the sets $\mathcal{D}_i$ of all the nodes j strongly influenced by i (upon which i strongly depends). The algorithm chooses the unassigned node i with minimal cardinality of $\mathcal{D}_i$ and forms a new aggregate containing i. Then it looks for the unassigned node j with the strongest negative connection with i, i.e. with the minimal value of L_{ij}: if $j \in \mathcal{D}_i$, then j is also added to the new aggregate together with i, otherwise i is left alone (in a single-node aggregate). The process is repeated, but the assigned nodes (i, and possibly j) are removed from the sets $\mathcal{D}_h$ they belong to, so that at each step the selected node is the one with the smallest number of *unassigned* strongly influenced nodes. This approach is therefore similar to those discussed in §4.1, except that the number of aggregations is not fixed a-priori, but depends on θ, and that all aggregations are performed "in parallel" (on the original matrix). Furthermore, because the number of aggregations can be too small, the *double* pairwise aggregation is also proposed, whereby the pairwise aggregation algorithm is run once providing a projector R_1. At that point, the projected matrix $L_1 = R_1 L R_1^T$ is formed, the pairwise aggregation algorithm is run again on L_1 providing a second projector R_2, and the final R is just the combination of the two $R = R_2 R_1$. The considered approach is even more similar to those in §4.1 and it is the one we tested in our computational experiments.

It has to be remarked that these techniques have been proposed for, and applied to, methods that are significantly different from the ones we consider. This is made apparent by the fact that the theoretical results are based on the estimate of the *multiplicative error* $\mathbf{e}$ of the current iterate $\mathbf{x}$, defined as the one which solves $A\ \mathrm{diag}(\mathbf{x})\mathbf{e} = 0$, as well as from the fact the matrix at the lower level is given by

$$RA\ \mathrm{diag}(\mathbf{x})R^T \quad ,$$

i.e. a scaling of the standard restricted system using the current iterate $\mathbf{x}$. This is motivated by the interpretation of the scaled matrix in Markov context: "for a link in the Markov chain from state i to state j, state i contributes to the probability of residing in state j in the steady state not just by the size of the transition probability from i to j but by the product of that transition probability and the probability of residing in state i" [20]. These modifications to the standard MG approach are not applicable in our setting (at the very least they require that $\mathbf{b} = \mathbf{0}$, which is not the case), nor they are any likely to be effective. Indeed [23] reports that "a standard Krylov acceleration technique cannot be applied, because the spaces involved are not related by a fixed preconditioner applied to residual vectors", while according to [24] "in the case of CG or GMRES acceleration of stationary multigrid cycles (our cycles are non-stationary), excellent convergence properties are often obtained because the spaces are nested". In order to obtain an acceleration, ad-hoc techniques are needed such as the minimization of a functional [23] or the solution of a two-dimensional quadratic programming problem and repeated use of W cycles [24]. This is true also for the *smoothed* version of aggregate operators proposed in [21] on the basis

of [10], which necessitate several ad-hoc interventions (a "lumping procedure" to keep the M-matrix nature of the coarse-level operators) and may exhibit high memory and execution time complexity. Our preliminary tests confirmed that smoothing applied to projectors almost always results in a substantially slower method. Thus, most of the theoretical results of [20, 21, 22, 23, 24] are largely inapplicable in our context, although the aggregation techniques can be mirrored.

4.3. Strength-based AMG operators

We remark that the projectors discussed in §4.2 are basically the "aggregation form" of non-aggregation operators, based on the notion of strong dependence, proposed for the Algebraic Multigrid Method (AMG) in [10, 11, 22, 23, 12]. In the AMG terminology, the projection process is denoted as finding the "C/F splitting", i.e. deciding which equations will remain in the coarsened (restricted) system (C) and which ones only belong to the finer (original) system (F). Once this is is done, the projector R is chosen so as to satisfy

$$(\mathbf{c}^T R)_i = \begin{cases} \mathbf{c}_i & \text{if } i \in C \\ \sum_{j \in C \cap \mathcal{S}_i} w_{ij} \mathbf{c}_j & \text{if } i \in F \end{cases}$$

where $\mathcal{S}_i$ denote the set of points that strongly influences i (i.e. the "opposite" of the sets $\mathcal{D}_i$ above), $\mathbf{c}$ is the coarse-level error approximation and the w_{ij}s are the *interpolation weights* given by [12]

$$w_{ij} = -\frac{L_{ij} + \sum_{m \in F \cap \mathcal{S}_i} \left(\frac{L_{im} L_{mj}}{\sum_{k \in C \cap \mathcal{S}_i} L_{mk}} \right)}{L_{ii} + \sum_{r \notin \mathcal{S}_i} L_{ir}} \quad .$$

While C and F are therefore used in rather different ways than constructing an aggregation operator, they are found in similar ways as the aggregations in §4.2. In particular, two heuristic criteria are defined to guide the search for the C/F splitting:

- for each F-point i, every point $j \in \mathcal{S}_i$ should either be in the coarse interpolatory set $C \cap \mathcal{S}_i$ or should strongly depend on at least one point in $C \cap \mathcal{S}_i$;
- the set of coarse points C should be a maximal subset of all points with the property that no C-point strongly depends an another C-point.

Satisfying both criteria is not always possible and typically the first has priority over the second if necessary. The standard heuristics used for finding the C/F splitting are two-phase greedy algorithms that closely resemble the neighborhood aggregation approach of §4.2, except for looking at the "directed" sets $\mathcal{S}_i$ and $\mathcal{D}_i$ rather than to the "undirected" $\mathcal{N}_i$ (as the pairwise aggregation does).

It is worth remarking that the results of [10, 11] were aimed at problematic cases in which "errors missed by standard relaxation processes can vary substantially along strong matrix connections". This suggested a generalization of the classical AMG so that a number of vectors (*prototypes*), which are related to the error components that relaxation cannot break down, are in the range of interpolation, instead of having only the vector $\mathbf{1}$ as prototype as usual in the AMG development. The matrices in our application are very close to those for which the original AMG was designed. Furthermore the aggregations operators based on strong connection of §4.2 worked quite well already. Thus, in our numerical tests, we employed the standard AMG approach [12], rather than the variants of [10, 11, 22, 23], which do not seem to fit our application. In our setting, these are interesting only as further examples of different possible definitions of strength-based operators. Nevertheless, there is no evidence that aggregation operators based on these principles should dramatically those considered in §4.2.

4.4. Combinatorial operators

While all the previous aggregation operators are inherently heuristic, the combinatorial operators studied in [45] are based on a sound theoretical analysis. Interestingly this is grounded on the theoretical results developed for the study of an apparently unrelated subject, that of *Steiner tree* and *Steiner support graph* preconditioners. The former were introduced in [37], while [42] and [43] extended the results to the latter.

"Sophisticated" support-graph preconditioners (with guarantees on the conditioning number of the resulting preconditioned system) can be constructed with a process that is based on identifying a partitioning of the graph

into "weakly interacting" clusters, selecting the (few) arcs that join them (which have to be "important" arcs, since their removal disconnects the graph), and possibly recursively repeating the process within each cluster [59, 44]. Since the clustering process is non-trivial to attain within low computational cost, the idea of Steiner support graph has been proposed in order to simplify it. Basically, once a set of promising vertex-disjoint clusters $\mathcal{V}_i$ is identified (with non-trivial algorithms which provide some appropriate guarantees on their quality), a new *Steiner node* r_i is created for each cluster, and the star trees rooted at r_i with leaves corresponding to the vertices in $\mathcal{V}_i$ are added to the preconditioner (which, of course, is defined for the new Steiner graph comprising all the original nodes plus the new roots). The process can then be repeated on the *quotient graph* corresponding to all the roots r_i; with appropriate care, this produces provably good preconditioners [43]. However in [45] it was realized that the very same process can be used for different purposes: namely the clusters $\mathcal{V}_i$ then become these of an aggregation operator, so that the quotient graph becomes the coarsened graph at the next level of a MGM. With appropriate choices, the guarantees on the conditioning obtained for the Steiner graph preconditioner can be "extended" to the coarsened system. This suggests the implementation of multigrid-like solvers. In particular, in [45] a solver running quite sophisticated cycles (more complex than the standard V and W cycles) and employing Jacobi as smoother obtains promising results for image processing applications. According to the authors, for the same underlying clustering approach, multigrid-like methods should almost always be preferable to the direct application of Steiner support graph preconditioners.

Because a sophisticated implementation of the approach of [45] was made available to us, we were able to test it on our instances. Our computational experience indicated that direct use of that approach did not seem to provide competitive results. This may be due to the use of "poor" smoothers like Jacobi, that are typically ineffective in our applications, as discussed in §2.4. However nothing prevents to re-use the "core" of the approach, i.e., the sophisticated choice of the aggregation operator, into a more standard MGM (accelerated or not) with more powerful Krylov-based smoothers.

5. Numerical tests

In this section we report a wide set of numerical tests aimed at comparing the computational behaviour of PCG and MG approaches on instances of (2), coming from real applications, in particular MCF problems. The stopping criterion used for all the methods was the slightly non-standard

$$|b_i - L_i\mathbf{x}| \leq \varepsilon \max(|b_i|, 1) \qquad \forall i \tag{10}$$

(with $\varepsilon =$ `1e-5`), because this is the form typically required by the specific application [32, 33]. Note that (10) is basically a (scaled) ∞-norm stopping criterion, and therefore more difficult to attain (for the same ε) than more standard 2-norm stopping criteria. The results have been obtained on a PC equipped with an Intel Core I5 750 CPU and 4 Gb of RAM running under Windows 7, using a 32-bits Matlab version 7.9 (R2009b).

5.1. Problem generators

The tests have been performed on matrices L coming from the solution of randomly-generated MCF instances. Three different well-known random problem generators have been used: `Net`, `Grid`, `Goto`. These have been used in several cases to produce (both single and multicommodity) flow test instances [13, 14, 30, 32, 33, 34]. Each generator produces matrices with different topological properties, as shown in Figure 2. Furthermore the solution of the MCF instances via a IP methods produces weight matrices Θ with a different behavior.

As the pictures show, the graph in `Net` problems has a random topological structure; these are the easiest instances to solve with the IP algorithm. Both `Grid` and `Goto` (Grid On TOrus) problems have a grid structure, but the latter are considerably more difficult to solve than the former, both in terms of IP algorithm and as the corresponding linear system. The difficulty of `Goto` is likely to be related by the structure of the L matrix, which is far from the block and the banded case. We recall that the latter is the classical pattern related to standard grid graphs. Under the same conditions (problem size, IP iteration and preconditioning), generally a `Goto` system requires an order of magnitude more PCG iterations than those required when solving the systems `Net` or `Grid`.

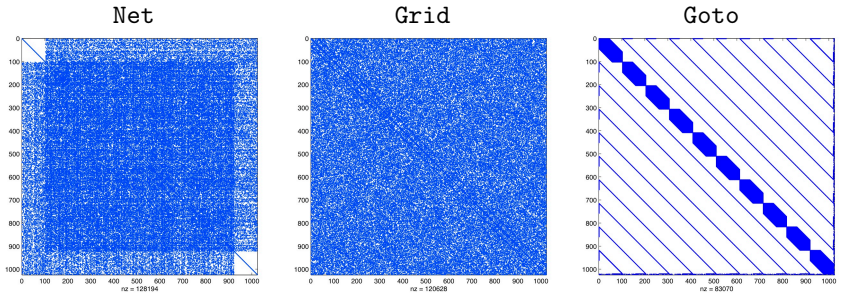

Figure 2: Structure of L for different problem classes.

5.2. List of methods

For our experiments, we have compared a large number of "pure" PCG approaches and accelerated multigrid (ACC-MG) approaches, i.e., the PCG algorithm where a single V-cycle is used as preconditioner. This is because on one hand PCG approaches have been popular and useful for the solution of this kind of systems in the context of MCF problems [13, 54, 14, 49, 32, 33] and on the other hand the most recent developments suggest that accelerated multigrid is both more effective and efficient than stand-alone multigrid [45, 23, 24]. In order to explore the multigrid strategy, we have taken into account several choices both for aggregation operators and preconditioners of PCG (as smoother in MGM).

The preconditioners employed are the diagonal one (D), the incomplete Cholesky one (IC) with tolerance parameter droptol (non-zero fill-in), the maximum spanning tree one (T), the "maximum spanning tree + diagonal" (T+D) one, obtained by summing to the T preconditioner the diagonal of the non-selected part the matrix. In particular, previous results [32, 33] have shown that the T+D preconditioner is always preferable to the T preconditioner for `Net` and `Grid` matrices, and therefore it is the only one employed in these cases. Conversely no dominance between T and T+D exist for the more difficult `Goto` matrices (with T+D being preferable in the first IP iterations and T in the last ones) and therefore both are tested in these cases.

All methods showed convergence, which however in some cases was exceedingly slow; thus we had to resort in setting an a priori upper-bound for the number of iterations. This was (somewhat arbitrarily) chosen as 1000 iterations for "pure" PCG and at 500 iterations for ACC-MG methods, since the latter have an higher cost per iteration.

We have to remark that providing consistent running times for all the methods proved to be very challenging, because of the different levels of optimizations of the available routines. For instance, while all aggregation operators have similar complexity, only for some of them efficient C implementations (and the corresponding MEX interface routines) were available, while others were implemented as ordinary (interpreted) m-files. This alone can produce orders-of-magnitude differences in running times. While highly sophisticated C++ implementations of support graph preconditioners are available [32, 33], for uniformity the built-in `pcg` function of Matlab has always been used, which, besides being an m-file, does not allow to exploit some of the relevant structure. Analogously the multigird approach has been implemented as an m-file and therefore is far less efficient than what a compiled version could be. Furthermore the maximum spanning tree has been computed with the `graphminspantree` function of Matlab; this uses an exact ordering rather than an approximated one as in [32, 33] and therefore is significantly less efficient, despite being a compiled C routine. Moreover it does not automatically produce the right column ordering which avoids fill-in in the Cholesky factorization of the preconditioner, thereby making this step less efficient, too. As a consequence, the running times in the following tables are not necessarily indicative of these that a fully optimized C or Fortran version may obtain. In particular, the times for forming and applying the projectors has been excluded from the figures, due to the above mentioned issues. Yet all methods have been implemented with the maximum possible uniformity, therefore we believe that the figures should be reasonably indicative of the relative performances of the approaches.

5.3. Preliminary comparisons

In a first phase of our experiments, we performed a tuning of the ACC-MG methods by comparing the different aggregation operators (cf. §4.1, 4.2 and 4.4) among them. Some data is shown in Table 1 for `Net` and `Grid` matrices and in Table 2 for `Goto` matrices. Only the case with $n = 2^{12}$ and density 64 ($m \approx 64n$) is

	D		IC		T+D		D		IC		T+D		D		IC		T+D	
IP	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
`Net`	CMG Aggregation						Max Aggregation						Minor Aggregation					
1	lev	lev	lev	lev	lev	lev	4	0.09	4	0.12	4	0.82	4	0.15	4	0.24	4	1.59
2	lev	lev	lev	lev	lev	lev	5	0.12	4	0.12	5	0.90	5	0.93	5	1.07	5	2.88
8	10	0.57	8	0.63	6	2.55	17	0.40	9	0.32	6	1.13	15	0.49	8	0.39	6	1.60
17	19	2.13	11	1.68	6	2.19	85	2.16	19	0.99	10	1.45	74	2.15	13	0.96	6	1.45
25	19	2.09	7	1.57	6	2.04	**	**	11	1.43	14	1.65	494	13.60	10	1.49	14	1.77
32	60	6.25	11	2.05	5	1.91	**	**	13	1.27	6	1.29	**	**	11	1.27	5	1.37
33	58	6.08	10	1.90	5	1.98	**	**	10	1.17	5	1.24	**	**	10	1.24	5	1.35
`Net`	Strength Aggregation						Neighborhood Aggregation						Double Pairwise Aggregation					
1	4	0.46	4	0.42	4	1.26	4	0.09	4	0.12	4	0.70	4	0.51	4	0.62	4	1.93
2	5	0.15	5	0.21	5	1.27	5	0.12	5	0.18	5	0.99	5	0.63	5	0.73	5	2.04
8	13	0.43	8	0.39	6	1.56	11	0.40	8	0.43	6	1.65	13	1.66	8	1.17	6	2.37
17	19	3.07	9	2.21	6	3.21	59	9.25	23	4.58	11	4.21	90	11.34	17	2.63	9	3.18
25	**	**	7	1.91	6	2.48	222	34.38	13	3.44	13	4.46	**	**	11	2.35	9	2.85
32	**	**	11	2.43	6	2.46	**	**	172	31.48	12	4.38	**	**	12	2.41	5	2.19
33	**	**	11	2.37	5	2.32	**	**	128	23.52	10	3.77	457	57.42	17	2.99	5	2.23
`Grid`	CMG Aggregation						Max Aggregation						Minor Aggregation					
1	lev	lev	lev	lev	lev	lev	4	0.11	4	0.12	4	0.87	4	0.71	4	0.76	5	2.65
2	lev	lev	lev	lev	lev	lev	7	0.15	7	0.20	7	0.99	5	0.17	5	0.21	5	1.32
11	lev	lev	lev	lev	lev	lev	5	0.12	6	0.18	5	0.93	5	0.84	5	0.98	5	2.91
23	14	1.57	6	1.37	7	2.26	73	1.82	12	1.11	19	1.85	30	0.93	7	1.12	6	1.51
34	18	1.95	12	1.65	7	2.27	145	3.65	21	1.04	17	1.74	86	4.43	13	0.87	8	1.49
44	18	1.90	42	6.27	5	1.88	**	**	146	6.59	14	1.67	**	**	166	7.86	4	1.31
45	18	1.95	41	6.09	5	1.91	**	**	55	3.34	11	1.54	**	**	**	**	6	1.40
`Grid`	Strength Aggregation						Neighborhood Aggregation						Double Pairwise Aggregation					
1	5	0.15	5	0.22	5	1.23	5	0.49	5	0.54	5	1.54	5	0.59	5	0.73	5	1.96
2	7	0.21	7	0.28	6	1.38	7	0.23	7	0.28	7	1.29	7	0.87	7	0.99	7	2.34
11	6	0.18	6	0.32	5	1.23	5	0.15	5	0.21	6	1.21	5	0.63	5	0.74	5	2.06
23	18	3.35	10	2.99	12	5.03	33	5.01	11	2.93	12	4.32	41	5.03	10	2.09	9	3.07
34	63	10.59	21	4.47	12	4.63	114	16.81	24	4.46	14	4.83	173	20.99	28	4.07	13	3.86
44	**	**	**	**	13	4.68	**	**	**	**	15	4.92	**	**	**	**	16	3.99
45	**	**	238	47.79	9	4.02	**	**	426	77.23	10	3.94	**	**	**	**	14	3.93

Table 1: ACC-MG methods: comparison between aggregation operators for `Net` and `Grid` 12-64, considering diagonal (D), incomplete Cholesky (IC), maximum spanning tree + diagonal (T+D) and maximum spanning tree (T) preconditioners.

shown, but the results are fairly indicative of the general trends. In the Tables, iteration numbers and total running times are reported for ACC-MG methods employing the different preconditioners. Here, "lev" indicates that the aggregation operator failed to produce a reasonable number of levels (see below for more details) or that there were memory issues with the matrices. For CMG Aggregation operators, for instance, this happens occasionally due to the fact that the available implementation still lacks an appropriate sparsification routine. Entries with "**" instead indicate that the approach could not converge to the required accuracy within the allotted iteration limit.

The Tables show a rather complex picture. There is no single aggregation operator that dominates all the others for all matrices, preconditioners and IP iterations. However, especially when paired with "powerful" preconditioners (IC, T+D), all aggregation operators are quite effective, often (although not always) performing similarly. Yet, it is important to remark that the different operators require different amounts of tuning. In particular, all strength-based aggregation operators strongly depend on the parameter θ. Finding an appropriate value of that parameter is not trivial and bad choices can lead to extremely poor performances. A few examples of this are shown in Table 3, where (a subset of) the levels produced on the same matrix by different settings of θ for the Strength Aggregation are reported. In the matrix, three stacked cells of the form $(k, \ldots, h)^T$ mean that $k - h + 1$ levels have been generated, the size of each one being just one less than that of the previous. Thus inappropriate setting of θ can lead to very many levels of very similar size, quite the opposite of the expected exponential reduction. Although the number of levels tends to decrease as θ does, the behavior is not monotone and can be highly erratic. Note that the AMG operators, although not aggregation ones, are still strength-based and suffer from the same drawbacks.

These issues happen much less frequently to the CMG operator, although sometimes the "descent" in the set of levels is not as smooth as one could imagine. For instance, for the `Grid` 16-8 matrix at IP 22 the obtained levels have size respectively 65535, 16907, 4518, 2461, 2198, 2142, 2122, 2114, 2110, 2104, 2097, 2092, 2087,

	D		IC		T+D		T		D		IC		T+D		T		D		IC		T+D		T	
IP	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
	CMG Aggregation								Max Aggregation								Minor Aggregation							
1	8	0.35	7	0.39	8	2.10	**	**	20	2.01	16	1.59	15	3.96	**	**	13	0.32	11	0.32	11	1.23	**	**
2	21	0.40	7	0.18	6	0.78	91	3.93	46	1.07	9	0.31	7	1.01	93	6.13	24	2.76	6	0.95	5	1.84	81	17.73
24	23	1.99	16	1.80	15	2.82	49	7.67	481	10.53	26	1.35	45	2.68	42	3.07	493	11.38	22	1.21	35	2.35	41	2.99
47	27	2.21	31	2.97	15	2.94	33	5.42	265	6.11	26	1.06	22	1.80	30	2.87	**	**	21	0.88	63	3.63	31	2.52
71	25	2.18	16	1.63	17	3.10	24	4.92	**	**	12	0.70	23	1.82	23	2.41	**	**	12	0.67	28	2.09	23	2.29
93	21	1.73	10	1.04	9	2.07	16	3.66	**	**	7	0.49	11	1.34	16	2.09	**	**	9	0.49	14	1.51	16	1.79
94	20	1.70	14	1.40	10	2.18	13	3.21	**	**	7	0.48	12	1.35	14	1.91	**	**	10	0.49	13	1.46	14	1.79
	Strength Aggregation								Neighborhood Aggregation								Double Pairwise Aggregation							
1	14	1.46	13	1.54	14	3.55	**	**	14	0.87	12	0.82	12	1.81	**	**	lev	lev	lev	lev	lev	lev	lev	lev
2	17	1.74	6	0.82	5	1.59	80	14.49	21	1.48	7	0.65	6	1.31	76	9.53	19	0.29	7	0.18	6	0.60	77	2.41
24	23	3.21	14	2.79	15	4.92	46	16.45	26	2.83	15	2.23	13	3.18	45	9.48	427	38.15	18	2.09	14	2.94	48	10.73
47	23	3.15	22	3.60	14	4.52	33	8.73	31	3.29	25	3.18	15	3.66	32	8.01	**	**	29	3.10	15	3.04	36	6.11
71	22	3.10	11	1.99	13	4.61	23	8.25	48	5.11	12	1.73	15	3.79	24	7.23	**	**	18	2.10	17	3.35	24	5.83
93	19	2.79	11	2.16	10	3.99	16	6.81	80	9.03	10	1.48	18	4.50	16	4.66	**	**	12	1.41	10	2.40	16	4.21
94	17	2.48	12	2.40	10	3.61	13	5.47	28	3.10	11	1.70	8	2.57	13	4.07	**	**	12	1.37	10	2.32	13	2.83

Table 2: ACC-MG methods: comparison between aggregation operators for `Goto` 12-64, considering diagonal (D), incomplete Cholesky (IC) and maximum spanning tree + diagonal (T+D) preconditioners.

	`Grid` 16-8, IP 11								
θ	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
	65535	65535	65535	65535	65535	65535	65535	65535	65535
	35959	34380	32660	30729	28578	26235	23610	20696	17113
	20919	18393	15959	13712	11562	9583	7687	5851	4085
	13133	10343	7984	6122	4777	3903	3292	2740	2296
	8755	6190	4333	3229	2627	2688	2434	882	1
	6336	4182	2637	2433	658	6	2	1	
	4818	3068	1456	11	1				
	4210	2964	362						
	4059	975	361						
	4022	974	360						
	⋮	⋮	⋮						
	1	1	1						
	`Net` 12-64, IP 1								
	4095	4095	4095	4095	4095	4095	4095	4095	4095
	1567	1567	1567	1567	523	523	523	523	523
	1322	1268	1123	796	450	404	332	152	63
	1290	1257	1111	689	449	403	331		
	1284	1255	1110	646	448	402	330		
	1283	1254	1109	645	447	401	329		
	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
	1	1	1	1	1	1	1		
	`Goto` 14-64, IP 1								
	16383	16383	16383	16383	16383	16383	16383	16383	16383
	7417	7418	7418	7404	1409	1394	1368	1288	1089
	3686	2914	2158	2080	912	864	757	674	579
	2283	2056	1621	1490	685	657	597	562	410
	1713	1752	1455	1363	642	581	586	551	405
	921	1576	1325	1344	285	242	559	43	397
	422	1292	1320	1320	284	241	558		396
	388	995	804	1319	283	240	557		395
	387	411	1	1318	282	239	556		394
	⋮	⋮		⋮	⋮	⋮	⋮		⋮
	378	408		1306	267	220	520		384
	1	1		2	1	1	1		1

Table 3: Multigrid levels as a function of θ for the Strength Aggregation.

	IP 2			IP 56			IP 84			IP 111		
droptol	it	p.t.	time	it	p.t.	time	it	p.t.	time	it	p.t.	time
1e-02	1581	0.23	18.42	**	**	**	**	**	**	**	**	**
1e-03	415	0.24	5.14	1916	0.45	25.95	**	**	**	**	**	**
1e-04	189	0.39	2.88	760	1.06	10.88	4198	0.50	57.12	**	**	**
1e-05	99	24.85	26.34	187	12.64	14.49	375	0.82	5.78	3875	0.45	48.60
1e-06	48	56.99	57.95	80	38.61	39.99	105	5.38	6.95	574	0.50	7.69
1e-07	28	0.90	108.54	46	54.67	55.64	51	27.25	28.09	95	0.72	1.98
1e-08	**	**	**	21	118.94	119.62	28	48.26	48.80	35	1.82	2.30
1e-09	**	**	**	**	**	**	17	67.39	67.82	18	13.65	13.94
1e-10	**	**	**	**	**	**	**	**	**	12	30.52	30.74

Table 4: Performances of IC PCG as a function of droptol.

2070, 2063, 2061, 2060, 2056, 2011, 1938, 1765, 926, and 1. By contrast, Max and Minor operators always precisely halve the number of nodes at each level, and therefore are preferable in this respect.

We also have to remark that the aggregation operators are not the only part of the approach which may require tuning. This is also true, in particular, for the Incomplete Cholesky preconditioner, which requires setting of the droptol parameter. Finding the appropriate setting is crucial for the overall performances, but this setting is by far not uniform across all our instances. This is shown e.g. in Table 4, where the performances of the IC PCG approach are reported for varying droptol for the same basic instance (`Goto` 14-64) and different IP iterations. In the Table, "it" is the number of PCG iterations, "time" the total time, and "p.t." the time spent in solving the preconditioned system at each iteration. As the Table shows, too large values of droptol result in "poor" preconditioners, which are very easy to invert but provide poor convergence and therefore high overall running time. On the contrary, too small values result in very dense preconditioners, which are effective but too costly. The right trade-off crucially depends on the IP iterations and can vary of several orders of magnitude.

5.4. Overall results

We proceed at comparing the best PCG variants, the best aggregation-based multigrid variants, and the standard AMG (with the same set of possible and powerful preconditioners). It is worth observing that our projectors are aimed at preserving the graph structure as much as possible, while in the standard AMG variants the projectors are chosen by using algebraic conditions on the entries of the considered matrices.

The results are shown in Table 5, Table 6, Table 7 for `Net`, `Grid`, `Goto` matrices respectively.

The tables show that accelerated multigrid approaches are in general competitive, when compared with PCG-based ones. Although the running times of the latter can be significantly smaller when the iteration count is reasonable (e.g., at low IP iteration for all networks and sizes with the D preconditioner), due to their much simpler structure and cost per iteration, the iteration count of ACC-MG is, with the same basic preconditioner, always considerably smaller. This is always true, and in particular for the `Goto` matrices, that are well-known to represent the most challenging structures [32, 33]. This results in the fact that ACC-MG has a considerably lower failure rate ("**" entries). Thus, PCG and graph-based projection operators do seem to have a good spectral complementarity.

However, the trade-off between efficiency and robustness is nontrivial. The only combination that regularly solves all the cases is ACC-MG with the diagonal preconditioner (D), that, as it could be expected, is sometimes not the most efficient one (cf. e.g. the `Net` matrices for large values of IP). That is, whenever they don't break down, "strong" preconditioners like IC and T+D usually significantly outperform D. Among them, IC can be very effective, especially in running time; however, this is partly due to the fact that the current implementation of the tree preconditioner is not completely optimized. Again, it also breaks down somewhat more often, and it is more sensitive to nontrivial appropriate choices of the fill-in parameter.

The comparison between AGO and AMG is also unclear. Usually they are quite comparable, but there are differences. AMG is more prone to failure especially for IP = 1, where all arcs are equal and it is more difficult to distinguish. For the same preconditioner, AGO and AMG trade blows in a complex pattern, that is very hard to predict; see for instance the behavior of AGO and AMG with T+D on the `Grid` 14-64 matrices for IP

	PCG						ACC-MG (AGO)						ACC-MG (AMG)					
IP	D		IC		T+D		D		IC		T+D		D		IC		T+D	
12-64	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	13	0.05	13	0.06	13	0.64	4	0.09	4	0.12	4	0.70	12	0.82	12	0.87	12	3.44
2	15	0.03	15	0.08	15	0.67	5	0.12	4	0.12	5	0.90	17	2.13	16	2.19	15	7.20
8	59	0.15	24	0.12	17	0.68	11	0.40	9	0.32	6	1.13	19	3.47	17	3.33	11	4.66
17	468	1.21	39	0.29	25	0.73	19	2.13	13	0.96	6	1.45	11	1.62	10	1.99	6	2.87
25	**	**	37	0.67	19	0.68	19	2.09	11	1.43	14	1.65	13	1.19	6	1.88	5	2.55
32	**	**	191	1.10	11	0.63	60	6.25	13	1.27	6	1.29	12	1.76	9	2.38	5	2.63
33	**	**	148	0.97	9	0.62	58	6.08	10	1.17	5	1.24	14	2.15	8	2.21	5	2.48
14-64	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	14	0.21	14	0.34	14	5.21	5	1.67	5	1.79	5	7.41	12	3.69	11	4.11	17	26.83
2	18	0.26	18	0.42	16	5.38	5	0.87	6	1.34	5	9.51	19	13.44	21	16.83	18	40.48
11	73	0.99	21	0.59	18	5.49	11	2.10	6	1.87	6	11.84	26	15.81	14	10.06	21	30.40
23	473	5.95	207	3.09	23	5.57	21	3.07	40	6.42	10	10.18	12	6.42	40	22.80	7	17.08
34	**	**	372	6.16	22	5.50	20	8.45	38	19.53	8	11.98	15	5.47	36	18.57	7	15.47
44	**	**	**	**	13	5.47	26	11.79	**	**	6	10.79	18	8.48	40	38.18	5	15.28
45	**	**	**	**	13	5.55	26	11.76	**	**	7	8.54	16	8.25	48	40.99	5	14.92
16-8	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	26	0.40	25	0.53	25	47.29	9	0.95	8	1.10	8	49.10	lev	lev	lev	lev	lev	lev
2	39	0.57	23	0.73	24	49.46	9	1.45	6	1.66	7	57.93	17	24.13	17	24.99	15	128.15
11	99	1.43	41	0.98	24	47.72	16	3.29	9	2.83	6	64.05	22	8.50	18	8.86	14	85.25
21	305	4.46	126	2.37	21	51.68	38	6.83	28	6.39	9	64.69	38	11.57	35	12.60	12	81.66
31	**	**	**	**	23	52.04	36	5.67	208	50.27	8	59.48	14	8.79	141	103.71	6	77.47
40	**	**	**	**	38	58.67	28	9.92	**	**	21	87.26	15	12.30	**	**	8	89.49
41	**	**	**	**	**	**	27	4.32	**	**	**	**	15	10.15	**	**	**	**

Table 5: Comparison between pure PCG, ACC-MG based on aggregation operators and ACC-MG based on algebraic multigrid approaches for `Net` matrices, considering diagonal (D), incomplete Cholesky (IC) and maximum spanning tree + diagonal (T+D) preconditioners.

	PCG						ACC-MG (AGO)						ACC-MG (AMG)					
IP	D		IC		T+D		D		IC		T+D		D		IC		T+D	
12-64	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	13	0.03	13	0.06	13	0.61	4	0.11	4	0.12	4	0.87	12	1.32	12	1.62	12	6.45
2	12	0.03	12	0.06	11	0.62	7	0.15	7	0.20	7	0.99	6	1.59	6	1.90	6	4.71
11	13	0.05	13	0.06	12	0.62	5	0.12	6	0.18	5	0.93	7	2.96	7	2.71	7	7.73
23	120	0.32	15	0.49	19	0.67	14	1.57	12	1.11	6	1.51	10	1.54	6	1.88	6	2.90
34	607	1.48	51	0.35	21	0.68	18	1.95	13	0.87	8	1.49	13	1.98	10	1.93	6	2.94
44	**	**	281	1.70	15	0.63	18	1.90	42	6.27	4	1.31	14	2.11	28	5.99	5	2.67
45	**	**	257	1.63	14	0.64	18	1.95	55	3.34	6	1.40	14	2.10	32	6.59	5	2.66
14-64	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	15	0.20	15	0.34	16	3.82	6	1.04	5	1.21	5	8.26	16	15.03	16	18.17	16	54.16
2	13	0.18	13	0.31	12	5.07	7	1.13	7	1.45	7	9.70	7	5.76	7	6.75	7	31.59
8	29	0.43	17	0.57	16	5.17	8	1.51	6	1.84	5	10.99	16	9.93	12	9.64	15	27.23
16	567	7.11	205	3.22	28	5.53	20	3.93	27	6.27	6	12.12	12	6.24	20	9.41	6	15.66
24	**	**	665	10.92	18	5.36	23	10.20	118	18.98	8	8.97	14	5.78	52	29.67	6	15.08
31	**	**	**	**	20	5.27	90	38.67	72	61.13	12	11.31	36	13.74	34	48.67	9	17.17
32	**	**	**	**	20	5.33	26	11.66	22	35.20	7	11.46	16	7.28	12	35.84	39	32.79
16-8	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	28	0.39	28	0.46	23	17.26	11	1.13	11	1.27	9	21.07	lev	lev	lev	lev	lev	lev
2	23	0.24	16	4.57	17	34.22	11	1.24	9	5.54	9	40.52	9	4.68	7	8.78	7	51.48
11	34	0.39	18	7.19	20	41.12	9	1.40	6	8.26	7	56.45	11	7.41	9	14.78	9	73.69
22	436	5.11	104	1.77	24	42.54	17	2.76	19	4.07	6	57.76	26	13.97	20	12.96	8	67.14
33	**	**	**	**	26	43.68	24	9.23	197	168.66	6	51.46	13	6.56	144	162.81	6	64.61
43	**	**	**	**	20	43.50	27	11.06	**	**	7	53.14	14	4.33	**	**	5	65.52
44	**	**	**	**	**	**	30	11.37	**	**	**	**	14	7.59	**	**	**	**

Table 6: Comparison between pure PCG, ACC-MG based on aggregation operators and ACC-MG based on algebraic multigrid approaches for `Grid` matrices, considering diagonal (D), incomplete Cholesky (IC) and maximum spanning tree + diagonal (T+D) preconditioners.

	PCG								ACC-MG (AGO)								ACC-MG (AMG)							
	D		IC		T+D		T		D		IC		T+D		T		D		IC		T+D		T	
12-64	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	30	0.07	27	0.07	26	0.51	807	3.86	13	0.32	11	0.32	11	1.23	**	**	17	1.38	15	1.40	14	2.63	**	**
2	475	1.03	68	0.18	43	0.53	94	0.82	19	0.29	7	0.18	6	0.60	77	2.41	7	1.06	5	0.96	3	1.95	59	14.67
24	**	**	417	1.21	472	2.73	69	0.82	23	1.99	22	1.21	35	2.35	41	2.99	15	1.81	12	2.10	10	2.96	43	12.51
47	**	**	829	1.96	898	4.80	54	0.74	27	2.21	21	0.88	22	1.80	31	2.52	18	2.02	22	2.91	12	3.21	31	6.77
71	**	**	381	0.95	**	**	41	0.70	25	2.18	12	0.67	23	1.82	23	2.29	16	1.88	14	1.96	12	3.36	25	7.28
93	**	**	367	0.90	**	**	26	0.67	21	1.73	7	0.49	11	1.34	16	1.79	16	1.82	12	1.65	8	2.46	16	4.58
94	**	**	398	0.96	**	**	23	0.63	20	1.70	7	0.48	12	1.35	14	1.79	15	1.71	11	1.57	9	2.74	13	4.35
14-64	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	54	0.62	54	0.71	51	3.79	**	**	15	3.99	16	4.74	11	15.70	**	**	55	5.92	56	6.24	63	15.75	**	**
2	**	**	189	2.88	279	7.97	37	3.08	46	4.61	14	3.18	27	7.64	21	7.90	22	2.83	8	2.82	10	6.50	20	10.01
28	**	**	384	22.94	**	**	137	6.25	36	3.69	14	19.37	21	7.78	112	22.43	20	2.96	12	20.93	14	9.95	99	30.17
56	**	**	760	10.88	**	**	102	6.27	34	3.29	17	3.57	20	8.41	78	18.03	19	2.69	13	4.13	16	10.62	58	22.86
84	**	**	375	5.78	**	**	73	5.63	35	3.35	10	2.40	18	7.76	50	14.43	22	2.99	8	2.93	14	10.20	47	20.43
111	**	**	574	7.69	**	**	35	4.63	39	5.11	9	2.69	27	9.32	19	8.34	18	2.35	10	2.49	11	8.89	20	10.88
112	**	**	91	1.85	**	**	34	4.44	39	3.93	8	1.98	19	8.31	19	8.89	19	2.49	5	2.10	11	8.86	19	10.37
16-8	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time	it	time
1	76	1.04	64	0.95	65	22.35	**	**	12	1.06	11	1.27	11	25.64	**	**	17	2.32	14	2.76	13	37.83	**	**
2	**	**	644	8.06	753	50.73	66	17.50	116	11.09	31	4.33	43	26.75	182	74.24	48	6.98	17	5.13	22	24.18	263	255.49
10	**	**	928	12.77	**	**	38	14.86	124	11.60	38	6.22	48	30.26	102	47.11	39	4.96	19	4.10	26	21.40	76	50.24
19	**	**	694	11.18	**	**	76	24.58	44	3.82	14	3.61	23	30.56	94	83.72	28	3.35	13	3.04	20	27.93	281	149.01
28	**	**	601	8.70	**	**	30	24.91	37	3.40	8	1.70	13	31.44	20	31.55	20	2.51	8	1.99	32	35.03	17	39.48
36	**	**	68	1.62	**	**	20	30.15	32	2.76	4	1.79	9	29.68	12	35.22	24	2.73	4	1.34	74	46.64	9	43.41
37	**	**	68	1.51	**	**	17	34.99	32	2.73	4	1.51	9	29.76	9	39.17	**	**	4	1.46	32	37.65	8	48.59

Table 7: Comparison between pure PCG, ACC-MG based on aggregation operators and ACC-MG based on algebraic multigrid approaches for `Goto` matrices, considering diagonal (D), incomplete Cholesky (IC), maximum spanning tree + diagonal (T+D) and maximum spanning tree (T) preconditioners.

31 and 32 (just two consecutive iterations). The general feeling is that more often than not the somewhat more sophisticated approaches underlying the AMG projection operators give them a slight edge upon the simple rules employed in the AGO ones; however, they also make them more prone to failure.

In summary, the Tables show that the aggregation-based multigrid variants present a favorable profile in terms of the efficiency to robustness trade-off. While not as effective as the pure PCG approach, it is also much more stable: indeed, if the parameters are not chosen properly (and this is sometimes not trivial), the PCG algorithm could become unacceptably slow. Simple aggregation rules can also be preferable to more sophisticated ones, as they result in comparable performances with less failure points.

What the Tables clearly show, however, it that the best choice of the combination between projection operator and smoother is still far from being completely understood. The relative performances of the approaches wildly vary among different network types and sizes, and even among consecutive IP iterations. We can therefore state that while the combination of graph-preserving aggregation operators and strong graph-based preconditioners seem very promising towards developing an optimal approach, more research is required if this goal has to be reached.

6. Conclusions

We have considered multi-iterative techniques of multigrid type for the numerical solution of large linear systems with (weighted) structure of graph Laplacian operators. We combined efficient coarser-grid operators with iterative techniques used as smoothers, showing that the most effective smoothers typically are of Krylov type with subgraph-based preconditioners, while the projectors have to be designed for maintaining as much as possible the graph structure of the projected matrix at the inner levels. We have developed necessary and sufficient conditions for this to happen, showing that, basically, all operators of this kind are based on node aggregations. These coarse-grid operators that preserve the fine-level structure are necessary to obtain good performances: see the case of the Poisson problem in Section 3.3 as a clear example and, conversely, the discussion in Section 2.5 about the fact that the same FWO is not satisfactory for general graphs, for which the structure of weighted Laplacian is not preserved. Interestingly enough, this framework is useful for explaining the reason why the classical projectors inherited from differential equations are good in the differential context and

why they behave unsatisfactorily in general for unstructured graphs. However, even the best graph-preserving projection operators are not sufficient to attain good performances in all cases, unless they are paired with "strong" smoothers.

Several numerical experiments have been conducted on different classes of matrices coming from Interior-Point methods for network flow problems. This is a challenging application because the matrices can have significantly different structure, and even for a fixed structure the weights can vary wildly between even consecutive IP iterations. Attaining optimal results in all cases is therefore particularly difficult, and robustness becomes of paramount importance, the "best" solution method being the one that performs acceptably in all cases. In this respect, combining graph-preserving projection operators and subgraph-based preconditioners appears to be a promising venue: the corresponding approaches seem to be reasonably uniformly effective irrespectively (at least to some degree) of the conditioning of the considered matrices, the type of the problems, and the IP iteration. This robustness is the most relevant result obtained in this paper, as while sophisticated PCG techniques exist that often work quite well, in some cases their convergence rate is exceedingly slow. Hence, our results significantly improve on the previous state of the art for this particular application.

However, our findings also show that several aspects still remain to be completely investigated. Our aim has been at methods that are not only robust but simple, both in terms of not requiring a particularly sophisticated implementation and, especially, in the sense of uniformly delivering good performances without the need of complex parameter tuning. In this respect, our simple coarse-grid operators are usually comparably effective as these based on more sophisticated ideas [45, 23, 24], while providing analogous or even better robustness with (or, probably, because of) less need of delicate adjustments of the algorithmic parameters. Yet, we still ended up with a pretty large set of possible algorithmic variants, among which choosing the best ones appears to be nontrivial.

Thus, one of the main conclusions that we can draw from these experiments is that the goal of developing a robust and efficient iterative approach for graph-structured linear systems is still rather far off. This is in particular true for our motivating application, i.e., KKT systems for IP methods for Linear (and, specifically, MCF) Problems, that are particularly challenging due to the high variability of the characteristics of the systems to be solved, even within the solution of the same LP. In this context there is no scope for any system-specific tuning, as the solver has to be capable of effectively and efficiently (enough) solving each system it is presented with. This is especially true as the solution of a LP is, in itself, most often not even the final goal: a (long) sequence of different but related LPs has to be solved in order to tackle even more difficult combinatorial problems. The existing methods, included those analyzed in this work, all require sophisticated implementations which involve many options and parameters: a-priori making the more efficient and robust choice is a challenging task, for which further investigations will be needed. In particular, we believe that it would be of interest to study the relationships between preconditioner and projector. This seems to be a largely unexplored path: some very preliminary results can be found in [26], but there is clearly still a lot to be done.

Several further research lines seems to be worth future investigation. From the theoretical viewpoint, it would be very interesting to obtain a rigorous characterization of the convergence speed of the proposed multigrid techniques with varying pre- and post-smoothers and choices of the projector. Regarding the latter, a better understanding of the effect of the projection step on the spectral properties of the matrices at lower level would be required in designing more effective approaches to choosing R, possibly simultaneously taking into account selection of an appropriate subgraph for preconditioning techniques.

A final important challenge would be the extension of our proposal to a non-symmetric setting, as it occurs when dealing with the celebrated Google problem [46, 25, 39], and to system matrices having structure of graph combined with structures of different nature, as it occurs when dealing with IP methods for problems related to more general graph-structured Linear or Quadratic Programs [7, 15, 16].

Acknowledgements

We are deeply indebted with Killian Miller and Ioannis Koutis for having shared with us their implementations of aggregation projectors, as well as for extremely useful discussion and support. We also thank the referees for the useful remarks and suggestions, which helped to improve the presentation of the paper. Finally, we acknowledge that the work of the third author has been partly supported via Donation **KAW 2013.0341** from the Knut & Alice Wallenberg Foundation, in collaboration with the Royal Swedish Academy of Sciences, supporting Swedish research in mathematics.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin. *Network flows: theory, algorithms and applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] A. Aricó, M. Donatelli, S. Serra-Capizzano *V-cycle optimal convergence for certain (multilevel) structured linearsystems.* SIAM J. Matrix Anal. Appl. 26(1), 186–214, 2004.

[3] O. Axelsson, M. Neytcheva. *The algebraic multilevel iteration methods – theory and applications.* Proc. of the 2nd Int. Coll. on Numerical Analysis, D. Bainov Ed., Plovdiv (Bulgaria), 13–23, 1993.

[4] M.S. Bazaraa, J.J. Jarvis, H.D. Sherali. *Linear programming and network flows.* Wiley, New York, NY, 1990.

[5] R. Bhatia. *Matrix Analysis.* Springer Verlag, New York, 1997.

[6] M. Bern, J. Gilbert, B. Hendrickson, N. Nuygen, S. Toledo. *Support-graph preconditioners.* SIAM J. Matrix Anal. & Appl. 27(4), 930–951, 2006.

[7] S. Bocanegra, J. Castro, A.R.L. Oliveira. *Improving an interior-point approach for large block-angular problems by hybrid preconditioners.* European Journal of Operational Research 231, 263–273, 2013.

[8] E.G. Boman, D. Chen, B. Hendrickson, S. Toledo. *Maximum-weight-basis preconditioners.* Numer. Linear Algebra Appl. 11(8-9), 695–721, 2004.

[9] E.G. Boman, B. Hendrickson. *Support theory for preconditioning.* SIAM J. Matrix Anal. Appl. 25(3), 694–717, 2003.

[10] M. Brezina, R.D. Falgout, S. MacLachlan, T.A. Manteuffel, S.F. McCormick, J. Ruge. *Adaptive smoothed aggregation (SA) multigrid.* SIAM Review 47(2), 317–346, 2005.

[11] M. Brezina, R.D. Falgout, S. MacLachlan, T.A. Manteuffel, S.F. McCormick, J. Ruge. *Adaptive algebraic multigrid.* SIAM J. Sci. Comput. 27(4), 1261–1286, 2006.

[12] W.L. Briggs, V.E. Henson, S.F. McCormick. *A multigrid tutorial.* SIAM, 2nd edition, 2000.

[13] J. Castro *A specialized interior-point algorithm for multicommodity network flows.* SIAM J. Opt. 10, 852–877, 2000.

[14] J. Castro, A. Frangioni. *A parallel implementation of an interior-point algorithm for multicommodity network flows.* in Vector and Parallel Processing – VECPAR 2000, J.M. Palma, J. Dongarra and V. Hernandez eds., Lecture Notes in Computer Science Vol. 1981, Springer-Verlag, 301–315, 2001.

[15] J. Castro, J. Cuesta. *Quadratic regularizations in an interior-point method for primal block-angular problems.* Mathematical Programming 130, 415–445, 2011.

[16] J. Castro, J. Cuesta. *Improving an interior-point algorithm for multicommodity flows by quadratic regularizations.* Networks 5, 117–131, 2012.

[17] A. Cayley. *A theorem on trees.* Quart. J. Math. 23, 376–378, 1889.

[18] D. Cherubini, A. Fanni, A. Frangioni, A. Mereu, C. Murgia, M.G. Scutellà, P. Zuddas. *A Linear Programming Model for Traffic Engineering in 100% Survivable Networks under combined IS-IS/OSPF and MPLS-TE Protocols.* Computers & Operations Research 38(12), 1805–1815, 2011.

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

[20] H. De Sterck, T.A. Manteuffel, S.F. McCormick, Q. Nguyen, J. Ruge. *Multilevel adaptive aggregation for Markov chains, with application to web ranking.* SIAM J. Sci. Comput. 30(5), 2235–2262, 2008.

[21] H. De Sterck, T.A. Manteuffel, S.F. McCormick, K. Miller, J. Pearson, J. Ruge, G. Sanders. *Smoothed aggregation multigrid for Markov chains.* SIAM J. Sci. Comput. 32(1), 40–61, 2010.

[22] H. De Sterck, T.A. Manteuffel, S.F. McCormick, K. Miller, J. Ruge, G. Sanders. *Algebraic multigrid for Markov chains.* SIAM J. Sci. Comput. 32(2), 544–562, 2010.

[23] H. De Sterck, T.A. Manteuffel, K. Miller, G. Sanders. *Top-level acceleration of adaptive algebraic multilevel methods for steady-state solution to Markov chains.* Advances in Computational Math. 35, 375–403, 2010.

[24] H. De Sterck, K. Miller, G. Sanders, M. Winlaw. *Recursively accelerated multilevel aggregation for Markov chains.* SIAM J. Sci. Comput. 32(3), 1652–1671, 2010.

[25] G. Del Corso, A. Gullí, F. Romani. *Fast PageRank computation via a sparse linear system.* Internet Math. 3(2), 259–281, 2005.

[26] P. Dell'Acqua, A. Frangioni, S. Serra-Capizzano. *Computational evaluation of multi-iterative approaches for solving graph-structured large linear systems.* Calcolo, 22, 10.1007/s10092-014-0123-y, 2015.

[27] M. Donatelli. *An algebraic generalization of local Fourier analysis for grid transfer operators in multigrid based on Toeplitz matrices.* Numer. Linear Algebra Appl. 17, 179–197, 2010.

[28] M. Donatelli. C. Garoni, C. Manni, S. Serra-Capizzano, H. Speleers. *Robust and optimal multi-iterative techniques for IgA Galerkin linear systems.* Computer Meth. Appl. Mech. Eng., http://dx.doi.org/10.1016/j.cma.2014.06.001, 2014.

[29] M. Donatelli, M. Semplice, S. Serra-Capizzano. *Analysis of Multigrid preconditioning for implicit PDE solvers for degenerate parabolic equations.* SIAM J. Matrix Anal. Appl., 32–4, 1125–1148, 2011.

[30] A. Frangioni, G. Gallo *A bundle type dual-ascent approach to linear multicommodity Min Cost Flow problems.* INFORMS J. Comput. 11, 370–393, 1999.

[31] A. Frangioni, B. Gendron *0-1 reformulations of the multicommodity capacitated network design problem.* Disc. Appl. Math. 157, 1229–1241, 2009.

[32] A. Frangioni, C. Gentile. *New Preconditioners for KKT Systems of Network Flow Problems.* SIAM J. Opt. 14, 894–913, 2004.

[33] A. Frangioni, C. Gentile. *Prim-based BCT preconditioners for Min-Cost Flow Problems.* Comput. Opt. Appl. 36, 271–287, 2007.

[34] A. Frangioni, A. Manca. *A Computational Study of Cost Reoptimization for Min Cost Flow Problems.* INFORMS J. On Comput. 18(1), 61–70, 2006.

[35] A. Frangioni, S. Serra-Capizzano. *Spectral analysis of (sequences of) graph matrices.* SIAM J. Matrix Anal. Appl. 23(2), 339–348, 2001.

[36] G.H. Golub, C.F. Van Loan. *Matrix computations.* North Oxford Academic, 1983.

[37] K. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems.* PhD Thesis, Carnegie Mellon University, CMU CS Tech Report CMU-CS-96-123, 1996.

[38] A. Greenbaum. *Analysis of a multigrid method as an iterative technique for solving linear systems.* SIAM J. Numerical Anal. 21(3), 473–485, 1984.

[39] R. Horn, S. Serra-Capizzano. *A general setting for the parametric Google matrix.* Internet Math. 3(4), 385–411, 2008.

[40] H.B. Keller. *Numerical methods for two-points boundary-value problems.* Blaisdell, London, 1968.

[41] G. Kirchhoff. *Uber die Auflosung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Strome gefuhrt wird.* Ann. Phys. Chem. 72, 497–508, 1847.

[42] I. Koutis. *Combinatorial and algebraic algorithms for optimal multilevel algorithms.* PhD Thesis, Carnegie Mellon University, CMU CS Tech Report CMU-CS-07-131, 2007.

[43] I. Koutis, G.L. Miller. *Graph partitioning into isolated, high conductance clusters: theory, computation and applications to preconditioning.* Symposiun on Parallel Algorithms and Architectures (SPAA), 2008.

[44] I. Koutis, G.L. Miller. *A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar Laplacians.* Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '07), 2007.

[45] I. Koutis, G.L. Miller, D. Tolliver. *Combinatorial Preconditioners and Multilevel Solvers for Problems in Computer Vision and Image Processing.* International Symposium of Visual Computing, 1067–1078, 2009.

[46] A. Langville, C. Meyer. *A survey of eigenvector methods for WEB information retrieval.* SIAM Review 47(1), 135–161, 2005.

[47] O.E. Livne, A. Brandt. *Lean Algebraic Multigrid (LAMG): Fast Graph Laplacian Linear Solver.* SIAM J. Sci. Comput. 34(4), 499–522, 2012.

[48] B. Mohar. *Some Applications of Laplace Eigenvalues of Graphs.* Graph Symmetry: Algebraic Methods and Applications, G. Hahn and G. Sabidussi eds., NATO ASI Ser. C 497, Kluwer, 225–275, 1997.

[49] R.D.C. Monteiro, J.W. O'Neal, T. Tsuchiya. *Uniform boundedness of a preconditioned normal matrix used in interior-point methods.* SIAM J. Opt. 15(1), 96–100, 2004.

[50] M. Ng, S. Serra-Capizzano, C. Tablino Possio. *Multigrid preconditioners for symmetric Sinc systems.* ANZIAM J. 45(E), 857–869, 2004.

[51] D. Noutsos, S. Serra-Capizzano, P. Vassalos. *The conditioning of FD matrix sequences coming from semi-elliptic Differential Equations.* Linear Algebra Appl. 428(2/3), 600–624, 2008.

[52] Y. Notay. *An aggregation-based algebraic multigrid method.* Electronic Trans. Num. An. 37, 123–146, 2010.

[53] R. Olfati-Saber, R.M. Murray. *Consensus problems in networks of agents with switching topology and time-dealays.* IEEE Trans. Automatic Control 49(9), 1520–1533, 2004.

[54] L.F. Portugal, M.G.C. Resende, G. Veiga, J.J. Jùdice. *A truncated primal-infeasible dual-feasible network interior point method* Networks 35, 91–108, 2000.

[55] J.W. Ruge, K. Stüben. *Algebraic multigrid.* in Multigrid methods, vol. 3 of Frontiers Appl. Math., SIAM, Philadelphia, 73–130, 1987.

[56] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS, Boston, 1996.

[57] S. Serra-Capizzano. *Multi-iterative methods.* Comput. Math. Appl. 26(4), 65–87, 1993.

[58] S. Serra-Capizzano, C. Tablino Possio. *Multigrid methods for multilevel circulant matrices.* SIAM J. Sci. Comput. 26(1), 55–85, 2004.

[59] D.A. Spielman, S.H. Teng. *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems.* Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 81–90, 2004.

[60] K. Stüben. *A review of algebraic multigrid.* J. Comput. Appl. Math. 128, 281–309, 2001.

[61] U. Trottenberg, C. Oosterlee, A. Schuller. *Multigrid.* Academic Press, 2001.

[62] P. M. Vaidya. *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners.* Unpublished manuscript. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, Minneapolis, 1991.

[63] P. Vanek, J. Mandel, M. Brezina. *Algebraic multigrid on unstructured meshes.* Technical Report X, Center for Computational Mathematics, Mathematics Department, 1994.

[64] R.S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, 1962.