Portland State University

PDXScholar

Mathematics and Statistics Faculty Publications and Presentations

Fariborz Maseeh Department of Mathematics and Statistics

2018

BootCMatch: A Software Package for Bootstrap AMG Based on GraphWeighted Matching

Pasqua D'Ambra
National Research Council, Napoli, Italy

Salvatore Filipone
Cranfield University, Cranfield, United Kingdom

Panayot S. Vassilevski
Portland State University, panayot@pdx.edu

Follow this and additional works at: https://pdxscholar.library.pdx.edu/mth_fac

 Part of the Mathematics Commons, and the Software Engineering Commons

Let us know how access to this document benefits you.

Citation Details

D'ambra, P., Filippone, S., & Vassilevski, P. S. (2018). BootCMatch: a software package for bootstrap AMG based on graph weighted matching. ACM Transactions on Mathematical Software (TOMS), 44(4), 39.

This Article is brought to you for free and open access. It has been accepted for inclusion in Mathematics and Statistics Faculty Publications and Presentations by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

BootCMatch: A Software Package for Bootstrap AMG Based on Graph Weighted Matching

PASQUA D'AMBRA, Institute for Applied Computing "Mauro Picone", National Research Council, Napoli, Italy
SALVATORE FILIPPONE, Centre for Computational Engineering Sciences, School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, United Kingdom
PANAYOT S. VASSILEVSKI, Department of Mathematics and Statistics, Portland State University and Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

This article has two main objectives: one is to describe some extensions of an adaptive Algebraic Multigrid (AMG) method of the form previously proposed by the first and third authors, and a second one is to present a new software framework, named *BootCMatch*, which implements all the components needed to build and apply the described adaptive AMG both as a stand-alone solver and as a preconditioner in a Krylov method. The adaptive AMG presented is meant to handle general symmetric and positive definite (SPD) sparse linear systems, without assuming any *a priori* information of the problem and its origin; the goal of adaptivity is to achieve a method with a prescribed convergence rate. The presented method exploits a general coarsening process based on aggregation of unknowns, obtained by a maximum weight matching in the adjacency graph of the system matrix. More specifically, a maximum product matching is employed to define an effective smoother subspace (complementary to the coarse space), a process referred to as compatible relaxation, at every level of the recursive two-level hierarchical AMG process.

Results on a large variety of test cases and comparisons with related work demonstrate the reliability and efficiency of the method and of the software.

CCS Concepts: • **Mathematics of computing → Solvers**;

Additional Key Words and Phrases: Algebraic multigrid, preconditioner, iterative solver, graph matching

ACM Reference format:
Pasqua D'Ambra, Salvatore Filippone, and Panayot S. Vassilevski. 2018. BootCMatch: A Software Package for Bootstrap AMG Based on Graph Weighted Matching. *ACM Trans. Math. Softw.* 44, 4, Article 39 (June 2018), 25 pages.
https://doi.org/10.1145/3190647

This work is supported in part by the NSF under Grant DMS-1619640. This work is supported in part by the EC under the Horizon 2020 Project *Energy oriented Centre of Excellence for computing applications (EoCoE)*, Project ID: 676629.
Authors' addresses: P. D'Ambra, Institute for Applied Computing "Mauro Picone," National Research Council, Via P. Castellino 111, Napoli, 80131, Italy; email: pasqua.dambra@cnr.it; S. Filippone, Centre for Computational Engineering Sciences, School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, MK43 0AL, United Kingdom; email: salvatore.filippone@cranfield.ac.uk; P. S. Vassilevski, Department of Mathematics and Statistics, Portland State University, Portland, Oregon, USA; email: panayot@pdx.edu, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA, USA.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 ACM 0098-3500/2018/06-ART39 $15.00
https://doi.org/10.1145/3190647

1 INTRODUCTION

Modern applications spanning a variety of fields from the discretization of Partial Differential Equations (PDEs) to discrete models on general networks very often require the solution of large and sparse linear systems of equations. The notion of "*large*" is qualitative and it expands, by orders of magnitude, over time; nowadays, it is not uncommon to encounter linear systems with hundreds of millions or even billions of unknowns.

Algebraic Multigrid Methods (AMGs) are a popular choice for solving various classes of such linear systems; their efficiency is based on the recursive application of a two-grid process consisting of a smoother, typically an iteration process that involves local updates, and a coarse-level correction. More specifically, the components of the error that cannot be handled by the smoother are reduced on a coarse level (or coarse grid); these components of the error are commonly referred to as *algebraically smooth* ones. In this respect, the multilevel solution process can be viewed as a *divide-and-conquer* algorithm working implicitly in a *frequency domain*: the components of the error that cannot be handled by the smoother are handled recursively by similar processes acting on coarser levels.

Smoother iterations are typically based on a simple relaxation method such as Jacobi or Gauss-Seidel. The coarse-grid correction consists of computing an approximate solution to the residual equation in an appropriately chosen coarse space, that is, solving a linear system of smaller size. The coarse space solution is then transferred back to the fine level by a suitable prolongation operator and this interpolant is used to update the current fine-level approximate solution. In AMG, the construction of the coarse space and of the prolongation operator linking it with the fine space is the main task in the set-up process, and it is referred to as the *coarsening algorithm*. The coarsening process either involves the system matrix, possibly with some additional information, or some assumptions are made about the origin of the matrix, possibly provided by the user.

Since the original appearance in the early 80s [12] and that of the popular variant in Reference [43], AMGs have relied on certain heuristic criteria, most notably the notion of *strength of connections*, which is employed in selecting the coarse-level variables, commonly referred to as coarse degrees of freedom. This has led to some very efficient and scalable parallel software libraries targeting classes of sparse matrices that correspond to discretizations of second-order diffusion type elliptic PDEs (see, for example, Reference [24]). Extensions to more general classes of matrices have been proposed; most notably, versions of AMG that led to the so-called adaptive or bootstrap AMG, including ones of aggregation type AMG [11, 16, 17]. More recently, many efforts were devoted to the design of adaptive cycles in bootstrap AMG for applications in lattice Quantum chromodynamics and Markov chains (see Reference [10] and the references herein). All of these approaches, in one way or another, exploit information gathered by testing the smoother and/or the current method to modify the method itself, so as to allow handling of the leftover components of the error. The basic concept goes back to the so-called *compatible relaxation* principle introduced by Achi Brandt in Reference [8], and further analyzed in References [25] and [26]. Our previous work in this field was motivated by the compatible relaxation and the adaptive AMG strategies. More specifically, we introduced in References [18] and [19] an algorithm named *coarsening based on compatible weighted matching*, which exploits a maximum product matching in the system matrix graph to enhance matrix diagonal dominance, reflecting the convergence properties of an appropriately defined compatible relaxation scheme. Information about the smooth error is then used to define edge weights assigned to the original matrix graph. In References [18] and [19], we proposed to use the coarsening based on compatible matching within a bootstrap process, which generates a composite AMG solver to obtain a method with a prescribed convergence rate.

In this article, we first introduce a modification to our original method for computing weights in the coarsening process and propose the use of an auction-type algorithm [6] for computing

near-optimal weighted matching with a reasonable computational cost. Our second objective is to describe the main features of our new C package—*BootCMatch: Bootstrap algebraic multigrid based on Compatible weighted Matching*; it comprises all the functionalities needed to build and apply our bootstrap AMG method exploiting the compatible matching coarsening algorithm described in the article.

The remainder of the article is organized as follows. In Section 2, we review the main concepts and preliminary information regarding two-level AMG, compatible relaxation, and graph matching. In Section 3, we give details on the *compatible weighted matching coarsening*, whereas in Section 4, we describe the main features of some available algorithms for computation of maximum weighted matching. Section 5 contains a description of the bootstrap algorithm, which employs coarsening based on matching. In Section 6, we introduce the software architecture of the *BootCMatch* code, describing its main data structure and functionalities. Section 7 discusses performance results on various test problems ranging from scalar and vector PDEs, and also including general symmetric and positive definite (SPD) matrices from the SuiteSparse Matrix Collection [20]. We conclude with additional remarks and future plans in Section 8.

2 PRELIMINARIES

2.1 Algebraic Framework of the Two-Level Hierarchical Method

We are concerned with the solution of linear systems of equations

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where $A \in \mathcal{R}^{n\times n}$ is a SPD matrix that is also large and sparse, using AMG based on aggregation. In this respect, we describe a two-level hierarchical method in which the original, fine-level, vector space $\mathcal{R}^n$ is decomposed as a direct sum of D-orthogonal components (for some given SPD matrix D)

$$\mathcal{R}^n = \mathcal{R}ange(P_c) \oplus^{\perp} \mathcal{R}ange(P_f). \tag{2}$$

The matrices $P_c \in \mathcal{R}^{n\times n_c}$ and $P_f \in \mathcal{R}^{n\times n_f}$, with $n = n_c + n_f$, are referred to as the *prolongator* and *complementary prolongator*, respectively. The ranges of the two prolongators correspond to the decomposition of the original fine space into a *coarse space* and a hierarchical *complementary space*. Exploiting the above decomposition, the matrix A admits the following two-by-two two-level hierarchical block form:

$$[P_c, P_f]^T A[P_c, P_f] = \begin{pmatrix} P_c^T A P_c & P_c^T A P_f \\ P_f^T A P_c & P_f^T A P_f \end{pmatrix} = \begin{pmatrix} A_c & A_{cf} \\ A_{fc} & A_f \end{pmatrix}. \tag{3}$$

In what follows, we refer to A_c as the *coarse matrix* whereas A_f is referred to as its *hierarchical complement*.

2.2 Compatible Relaxation

We consider smoothers (relaxation schemes) of the Richardson type:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k), \tag{4}$$

assuming that $(M + M^T - A)$ is SPD, which is equivalent to the fact that the smoother is convergent in the A-norm (e.g., Reference [47]). Note that the error propagation matrix corresponding to the iteration in Equation (4) is $(I - M^{-1}A)$.

A *compatible relaxation* is defined in Reference [8] as a smoother capable of keeping the coarse variables invariant; such a smoother may be used to define practical algorithms for selecting coarse spaces in AMG [8, 9, 15, 25, 34]. In particular, the convergence rate of a compatible relaxation may

be used to estimate the quality of the coarse grid, i.e., the ability to represent algebraically smooth error components.

Starting from the decomposition in Equation (2), we can see that the modified relaxation method represented by the following error propagation:

$$\mathbf{e}_{k+1} = \left(I - P_f\left(P_f^T M P_f\right)^{-1} P_f^T A\right)\mathbf{e}_k$$

is a compatible relaxation. The coarse variables can be defined as $\mathbf{w}_c = \pi_c \mathbf{w}$, where $\pi_c = P_c(P_c^T D P_c)^{-1}(P_c)^T D\mathbf{w}$ (i.e., as the D-orthogonal projection of $\mathbf{w}$ onto the coarse space); by construction, we have $(P_c)^T D P_f = 0$ and therefore $\pi_c P_f = 0$. This implies that $\pi_c \mathbf{e}_{k+1} = \pi_c \mathbf{e}_k$, i.e., the coarse variables are invariant under this iteration. In general, if M_f is a well-defined smoother for the hierarchical complement A_f, we can think of it as being derived from a global smoother, i.e., $M_f = P_f^T M P_f$; it can act as a compatible relaxation for the matrix in Equation (3), and the complementary space $\mathcal{R}ange(P_f)$ can be defined as the *effective smoother space*, that is, the space on which the smoother must be effective.

In Reference [25] (Theorem 5.1), it has been proven that the quantity $\rho_f = \|I - M_f^{-1} A_f\|_{A_f}$ is an upper bound of a measure of the quality of the coarse matrix A_c: if the compatible relaxation converges fast, the smoother is able to damp large eigenmodes of the matrix A, whereas the coarse matrix A_c represents well the corresponding smooth vectors.

2.3 Graph Matching

Let $G = (V, E)$ be the undirected adjacency graph of the symmetric matrix $A = (a_{ij})_{i,j=1,\ldots,n}$, where the vertex set V consists of the row/column indices of A and the edge set E corresponds to the sparsity pattern of the matrix A so that $(i, j) \in E$ iff $a_{ij} \neq 0$. A *matching* in G is a subset of edges $\mathcal{M} \subseteq E$ such that no two edges share a vertex. The number of edges in $\mathcal{M}$ is the *cardinality* of the matching; a maximum cardinality matching is a matching which includes the largest possible number of edges, and a matching is *perfect* if its edges touch all vertices. In the following, we assume that we have a set of weights $c_{ij} \geq 0$ associated with the edges; the *weight of a matching* is then defined as:

$$C(\mathcal{M}) = \sum_{(i,j)\in\mathcal{M}} c_{ij}. \tag{5}$$

A *maximum product matching* in the graph of a sparse matrix is a matching that maximizes the product of the entries a_{ij} for each edge (i, j) in the matching $\mathcal{M}$; maximum product matchings are successfully used in sparse matrix direct solvers to move large matrix entries onto the main matrix diagonal (see References [22, 23, 27, 30, 31]). For general weights c_{ij} associated with the edges, we are interested in a *maximum product weighted matching*, that is, one which maximizes the product of the weights of all edges in the matching.

It has been shown (see Reference [22]) that maximizing the product $\prod_{(i,j)\in\mathcal{M}} c_{ij}$ is equivalent to maximizing the quantity:

$$\sum_{(i,j)\in\mathcal{M}} (\log|c_{ij}| - \log max_i|c_{ij}|),\ c_{ij} \neq 0. \tag{6}$$

In the following, we describe how we use a maximum product weighted matching of the matrix graph, defining edge weights, which depend on a general (smooth) vector and the given matrix A, to generate a pairwise aggregation of unknowns (associated with the vertices of the matrix graph) for use in the AMG coarsening process. Note that using graph matching for aggregation-based AMG was already proposed in other works, e.g., References [14] and [13].

3 COMPATIBLE RELAXATION AND MAXIMUM WEIGHTED MATCHING

Let $\mathcal{M} = \{e_1, \ldots, e_{n_p}\}$ be a matching of the graph G, with n_p the number of index pairs, and let $\mathbf{w} = (w_i)$ be a given (smooth) vector; for each edge $e = (i, j)$, we define two local vectors:

$$\mathbf{w}_e = \frac{1}{\sqrt{w_i^2 + w_j^2}} \begin{bmatrix} w_i \\ w_j \end{bmatrix} \tag{7}$$

and

$$\mathbf{w}_e^{\perp} = \frac{1}{\sqrt{w_j^2/a_{ii} + w_i^2/a_{jj}}} \begin{bmatrix} -w_j/a_{ii} \\ w_i/a_{jj} \end{bmatrix}, \tag{8}$$

where $D_e = \left[\begin{smallmatrix} a_{ii} & 0 \\ 0 & a_{jj} \end{smallmatrix}\right]$ is the diagonal of the local matrix, the restriction of A to the edge e. Note that by construction, we have $\mathbf{w}_e^T D_e \mathbf{w}_e^{\perp} = 0$. Based on the above vectors, we can form the following block-diagonal matrices:

$$\tilde{P}_c = \text{blockdiag}(\mathbf{w}_{e_1}, \ldots, \mathbf{w}_{e_{np}}), \qquad \tilde{P}_f = \text{blockdiag}(\mathbf{w}_{e_1}^{\perp}, \ldots, \mathbf{w}_{e_{np}}^{\perp}),$$

from which two corresponding *prolongators* can be constructed:

$$P_c = \begin{pmatrix} \tilde{P}_c & 0 \\ 0 & W \end{pmatrix} \in \mathcal{R}^{n \times n_c}, \qquad P_f = \begin{pmatrix} \tilde{P}_f \\ 0 \end{pmatrix} \in \mathcal{R}^{n \times n_p}. \tag{9}$$

In Equation (9), W is the diagonal matrix

$$W = \text{diag}(w_k/|w_k|), \quad k = 1, \ldots, n_s,$$

which has been considered for possible self-matched or unmatched nodes and $n_c = n_p + n_s$.

Let $D = \text{diag}\,(A)$ be the diagonal matrix having the same diagonal entries of A. The matrix P_c represents a piecewise constant interpolation operator whose range includes the original (smooth) vector $\mathbf{w}$; furthermore, by construction, $\mathcal{R}ange(P_c)$ and $\mathcal{R}ange(P_f)$ are orthogonal with respect to the D-inner product on $\mathcal{R}^n$, i.e., the property in Equation (2) holds.

A two-by-two block form of as in Equation (3) is easily defined for the matrix A by reordering the unknowns according to the index pairs in the matching $\mathcal{M}$, and moving all the unknowns corresponding to unpaired vertices at the bottom.

In the above setting, each relaxation scheme well defined for the block A_f is a compatible relaxation; therefore, our goal is to seek a matching $\mathcal{M}$ which makes A_f as well conditioned, or as more diagonally dominant, as possible. In this way, the compatible relaxation becomes effective on the space $\mathcal{R}ange(P_f)$ and $\mathcal{R}ange(P_c)$ would be considered a good coarse space.

To this end, we choose $\mathcal{M}$ so that the product of the diagonal entries of A_f is as large as possible. Note that the diagonal of A_f is a subset of the entries of a new matrix $\widehat{A} = (\widehat{a}_{ij})$, a function of the original system matrix A, and of the vector $\mathbf{w}$, whose entries are defined as follows:

$$\begin{aligned} \widehat{a}_{ij} &= \frac{1}{w_j^2/a_{ii} + w_i^2/a_{jj}} \begin{bmatrix} -w_j/a_{ii} \\ w_i/a_{jj} \end{bmatrix}^T \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \begin{bmatrix} -w_j/a_{ii} \\ w_i/a_{jj} \end{bmatrix} \\ &= \frac{1}{a_{jj}w_j^2 + a_{ii}w_i^2} \left(a_{jj}w_j^2 + a_{ii}w_i^2 - 2a_{ij}w_i w_j \right) \\ &= 1 - \frac{2a_{ij}w_i w_j}{a_{ii}w_i^2 + a_{jj}w_j^2}. \end{aligned} \tag{10}$$

The matrix $\widehat{A}$ has the same adjacency graph as that of A, and the computational cost for building it is $O(nnz)$, where nnz is the number of non-zeros of A, or equivalently the size of the edge set E. Therefore, we can use $\widehat{A}$ as a viable weight matrix for the edge set E to obtain a pairwise aggregation

of the original vertex set, driven by a maximum product weighted matching for the graph G; the entries of $\widehat{A}$ serve as edge weights, leading to a matrix A_f in Equation (3) exhibiting (generalized) diagonal dominance.

Note that the choice of the weights in Equation (10) is a modification with respect to the weights we originally proposed in Reference [18], due to the use of the D-inner product defined on $\mathcal{R}^n$. We introduce this new product since our goal is to make A and the diagonal of A as spectrally equivalent as possible in the subspace complementary to the coarse space; therefore, it is natural to consider the D-orthogonal complement to the coarse-space. In our experiments, we verified that the use of the corresponding new weights for the matrix graph leads to aggregations based on the maximum product matching, which generally improve convergence of our final AMG.

Our basic pairwise aggregation procedure based on weighted matching is detailed in Algorithm 1. It builds a partition $G_k,\ k = 1, \ldots,\ n_c$ of the vertex set $V = 1, \ldots, n$, where each aggregate G_k is normally a pair of matched indices. In the case of self-matched indices or possible unmatched indices due to sub-optimal solutions, we may obtain a partition containing singletons.

ALGORITHM 1: *Pairwise aggregation based on maximum product matching*

Data: graph G of dimension n, weight matrix $\widehat{A}$
Result: n_p, n_s, n_c and sets of aggregates $G_1, \ldots G_{n_c}$

- Compute $\mathcal{M}$ **maximum product matching** for the weighted graph $(G, \widehat{A})$.
- Initialize: $n_c = 0$, $n_p = 0$, $n_s = 0$;
  $U = [1, \ldots, n]$;
- **while** $U \neq \emptyset$ **do**

Pick $i \in U$;
if $\exists j \in U \setminus \{i\}$ *such that* $(i, j) \in \mathcal{M}$ **then**
  $n_p = n_p + 1$;
  $n_c = n_c + 1$;
  $G_{n_c} = \{i, j\}$;
  $U = U \setminus \{i, j\}$;
else
  $n_s = n_s + 1$;
  $n_c = n_c + 1$;
  $G_{n_c} = \{i\}$;
  $U = U \setminus \{i\}$
end
  end

Algorithm 1 is an automatic aggregation procedure once the matching is constructed, in the sense that it only uses the matrix entries with no additional assumptions such as strength of connection measures. The aggregation algorithm is the key element of the coarsening scheme described in this section; by applying it in a recursive manner, we can build a multilevel hierarchy of coarse spaces and matrices, as described in Algorithm 2.

We observe that, during the coarsening process, the smooth vector components corresponding to some index set of the partition could become very small, i.e., these error components are sufficiently reduced by the smoother. In these cases, we associate the related unknowns to the vector space $\mathcal{R}ange(P_f)$. In details, if $(i, j) \in \mathcal{M}$ is such that $\sqrt{w_i^2 + w_j^2} < TOL$, we consider the corresponding indices as unpaired; furthermore, for each index i such that $|w_i| < TOL$, we consider i as

ALGORITHM 2: *Coarsening based on compatible weighted matching*

Data: matrix A, (smooth) vector $\mathbf{w}$, maximum dimension for the coarsest matrix *maxsize*, maximum number of levels *maxlev*
Result: hierarchy of coarse matrices A^k (and prolongators)
$A^1 = A$, $k = 1$;
while *dim*$(A^k) >$ maxsize *and* $k <$ maxlev **do**
  build $\widehat{A}^k$ from A^k and $\mathbf{w}^k$;
  compute partition G_l^k by **Algorithm 1** for graph associated with A^k and the weight matrix $\widehat{A}^k$;
  build P_c^k and A_c^k from G_l^k, A_k and $\mathbf{w}^k$;
  $A^{k+1} = A_c^k$, $\mathbf{w}^{k+1} = (P_c^k)^T \mathbf{w}^k$;
  $k = k + 1$;
end

only fine-grid index and modify the complementary prolongator in Equation (9) as in the following:

$$P_f = \begin{pmatrix} \tilde{P}_f & 0 \\ 0 & I \end{pmatrix} \in \mathbb{R}^{n \times (n_p + n_f)}, \tag{11}$$

where $I \in \mathbb{R}^{n_f \times n_f}$ is the identity matrix and n_f is the number of *only fine-grid indices*. In our experiments, we choose TOL as the machine epsilon.

To obtain a more aggressive coarsening strategy, we can compose in a multiplicative framework the prolongators obtained by the pairwise aggregation of Algorithm 2; for example, starting from nl prolongators, with nl number of levels of the AMG hierarchy built with the basic pairwise aggregation of unknowns, we can define a new hierarchy with $\lceil nl/2 \rceil$ levels, corresponding to aggregates of size at most $m = 2^2$, by computing multiple products each involving two consecutive prolongators. This scheme may be extended to aggregates of size at most $m = 2^s$, for some suitable value s, by computing the product of s consecutive pairwise prolongators.

As already observed, the prolongation operator P_c we used for moving between the selected coarse space and the original fine space corresponds to a piecewise constant interpolant. To improve performance of a single V-cycle, we also consider the use of a more accurate interpolation operator obtained by applying one step of a weighted-Jacobi smoother to the basic piecewise constant interpolation, as in the smoothed aggregation AMG introduced in Reference [46]. The actual prolongators $\hat{P^k}$ are obtained from P^k as in the following:

$$\hat{P^k} = (I - \omega D^{-1} A^k) P^k,$$

where $D = \text{diag}(A^k)$ and $\omega \approx 1/\rho(D^{-1}A^k)$; to estimate the spectral radius $\rho(D^{-1}A^k)$, we use the infinity norm of $D^{-1}A^k$.

4 ALGORITHMS FOR MAXIMUM WEIGHTED MATCHING

Many algorithms have been devised to compute maximum weighted matchings; typically, they search first for a maximum cardinality matching and then maximize the weight of the matching.

One of the most widely used algorithms for computing a maximum cardinality maximum weight matching problem, which is also known as the *assignment problem*, is that implemented in the subroutine MC64 available from the HSL library [45]. MC64 works on bipartite graphs, that is, graphs where the vertex set is partitioned into two subsets V_r and V_c (for example, the rows and the columns of A), such that $(i, j) \in E$ connects $i \in V_r$ and $j \in V_c$ [21]; it uses the Hungarian algorithm [32] to search optimal augmenting paths in the matrix between unmatched vertices. MC64

has a worst-case computational complexity of $O(n(n + nnz) \log n)$, which makes it very expensive to apply in the context of Algorithm 1. Moreover, the Hungarian algorithm is a sequential process; therefore, it represents a roadblock in the search for a parallel implementation of Algorithm 1.

If we relax the optimality requirement and settle for a large but not necessarily maximum product of the weights, we can use various approximate algorithms giving near-optimal results in cardinality and weight with better computational complexity and offering improved opportunities for parallelism; we tested two such algorithms—a half-approximation algorithm [41] and an auction-type algorithm [6].

4.1 Half-Approximate Matching

The half-approximate algorithm we implemented in our code is a greedy algorithm, which is able to find, in linear time $O(nnz)$, a matching whose total weight is at least half the optimal weight and whose cardinality is at least half the maximum cardinality.

Algorithm 3, proposed by Preis [41], uses a queue-based mechanism; at each iteration, it adds to the matching a local dominant edge, i.e., an edge (i, j) with weight c_{ij}, such that $\arg\max_k c_{ik} = \arg\max_k c_{jk} = c_{ij}$; the matched edge and its vertices are removed and the resulting reduced graph is considered for the next iteration until the edge set is empty.

Note that our interest is in the maximum product weighted matching, while the approximation algorithm computes a matching to maximize the total weight; therefore, we apply the transformation in Equation (6) to the assigned edge weights.

We note that some parallel implementations of this half-approximate matching algorithm have been proposed [28, 40].

ALGORITHM 3: *Algorithm for half-approximate matching*

Data: $G = (V, E)$ graph, C edges weight matrix
Result: matching $\mathcal{M}$
$\mathcal{M} = \emptyset$;
while $E \neq \emptyset$ **do**
  take a locally dominant edge $(i, j) \in E$, i.e., such that $\arg\max_k c_{ik} = \arg\max_k c_{jk} = c_{ij}$;
  add $(i, j) \in \mathcal{M}$;
  remove all edges incident to i and j from E;
end

4.2 Auction-Type Algorithm

A different algorithm to compute a near-optimal maximum weight maximum cardinality matching is the auction algorithm first proposed by Bertsekas [5]. In our code, we use the implementation described in Reference [31] and available in the Sparse Parallel Robust Algorithms Library (SPRAL) [29].

The original auction algorithm works on a bipartite graph where matrix rows represent buyers and matrix columns represent objects. Each edge weight c_{ij} represents the benefit to buyer i of obtaining the object j. A standard auction algorithm defines some row variables u_i, called the cost variables and initialized to zero, and looks for the row index i such that $i = \arg\max_k(c_{kj} - u_k)$, for each unmatched column j. Any column j for which $c_{ij} - u_i > 0$ is a candidate for matching with row i, but only the column index j_i with the largest value is chosen, while the others remain unmatched. The row variable u_i is then updated with the value $u_i = u_i + (c_{ij_i} - u_i) - \max_{k \neq i}(c_{kj_i} - u_k)$, i.e., the variable u_i is updated to be the cost of using the second best row; in other words, u_i

is the reduction in the objective if j_i is not matched to i. A parameter ϵ is generally added to the variable u_i to guarantee a minimum increase in the objective function at each iteration. The choice of the initial value for parameter ϵ and the way to scale it during the iterations are important for the optimality and convergence properties of the algorithm; these issues are discussed at length in Reference [44]. The number of iterations to convergence increases for decreasing values of ϵ, but on the other hand, large values of ϵ may produce matchings with small weight. The auction algorithm implemented in SPRAL starts from a constant value ($\epsilon = 0.01$) and then applies an increasing scaling strategy in which the value of ϵ is chosen as $\min(1.0, \epsilon + 1/(n+1))$ where n is the size of the vertex set. We used the same choices in our experiments.

The original auction algorithm has a worst-case computational complexity of $O(n \cdot nnz \log(Cn))$, in the case of integer weights, where $C = \max_{ij} |c_{ij}|$. However, in our application, we use a version of the sequential auction algorithm as implemented in SPRAL and described in Reference [31] (see Algorithm 4), where the first column j for which $c_{ij} - u_i > 0$ is matched to i, without looking for the best column; then, if there already exists k such that $(i,k) \in \mathcal{M}$, this last edge is removed from the matching $\mathcal{M}$ and the column k becomes unmatched. This version reduces the cost per iteration and the average number of iterations; as observed in our experiments, it produces good quality aggregates and leads to effective aggregation schemes at a much reduced cost compared to the use of the optimal (but expensive) algorithm MC64.

As in the half-approximate matching, the auction algorithm computes a matching with a maximum weight; therefore, to get a maximum product matching an edge weight transformation is needed. In the SPRAL code, the following transformation is applied:

$$\bar{c}_{ij} = \alpha + \log |c_{ij}| + (\alpha - c_j),$$

where $c_j = \max_i \log |c_{ij}|$ and $\alpha = \max_{ij}(c_j - \log |c_{ij}|)$. Introducing α reflects the fact that the matching objective includes not only the maximum weight but also the maximum cardinality (see Reference [31]).

ALGORITHM 4: *Simplified Auction Algorithm*

Data: $G = (V_r, V_c, E)$ bipartite graph of dimension n, $C = (c_{ij})$ edges weights, *maxiter*
Result: matching $\mathcal{M}$
Initialize $\mathcal{M} = \emptyset$, $U = \emptyset$, $\mathbf{u} = \mathbf{0}$, $\epsilon = 0.01$;
while $|\mathcal{M}| < n$ *and* $|\mathcal{M}|$ *is changing and iter* $<$ *maxiter* **do**
  scale $\epsilon = \min(1.0, \epsilon + 1/(n+1))$;
  for $j \in V_c \setminus U$ *and unmatched* **do**
    find $i = \arg\max_k (c_{kj} - u_k)$, $pval = c_{ij} - u_i$, $qval = \max_{k \neq i}(c_{kj} - u_k)$;
    if $pval > 0$ **then**
      $u_i = u_i + pval - qval + \epsilon$;
      add $(i,j) \in \mathcal{M}$;
      if $(i,k) \in \mathcal{M}$ **then**
        add $k \in U$;
        remove (i,k) from $\mathcal{M}$;
      end
    else
      add $j \in U$;
    end
  end
end

We also note that an efficient parallel algorithm for auction-based weighted matching has been proposed in Reference [44].

5 BOOTSTRAP AMG BASED ON COMPATIBLE WEIGHTED MATCHING

One of the main ingredients of Algorithm 2 is the choice of the initial vector $\mathbf{w}$, which is meant to be a good approximation to an algebraically smooth vector for the problem at hand. As already mentioned, following the main concept of bootstrap/adaptive AMG developed in References [11, 16, 17], in Reference [18], we proposed to use *coarsening based on compatible weighted matching* within a similar adaptive process. The goal is to improve the convergence rate of the final AMG scheme by generating algebraically smooth vectors obtained dynamically throughout the adaptive set-up process, by testing the most recent method for possible further improvement.

In our bootstrap process, we build a solver B composed from a number of different AMG cycles. The error propagation matrix corresponding to the operator B has the following product form:

$$I - B^{-1}A = (I - B_\ell^{-1}A) \dots \ (I - B_1^{-1}A)(I - B_0^{-1}A), \tag{12}$$

where each B_r is an AMG cycle operator built with its own hierarchy of aggregates. These are constructed by a weighted matching of the original matrix graph. The weights come from the algebraically smooth vector $\mathbf{w}_r$ computed by testing (in an iteration procedure) of the currently available (up to $r-1$) composite solver, i.e., the solver composed from the previously constructed AMG operators $B_0, \dots, B_{r-1}$.

More specifically, the composite solver is constructed as follows. Starting from a given general (smooth) vector $\mathbf{w}_0$, this vector is used to build an initial AMG hierarchy by Algorithm 2. This AMG hierarchy is used to define an initial AMG operator B_0. Then, we use the operator as a solver applied to the homogeneous system $A\mathbf{w} = \mathbf{0}$, starting with a non-zero random initial iterate by successively computing $\mathbf{w}^k := (I - B_0^{-1}A)\mathbf{w}^{k-1}$ for a fixed number of iterations $k = 1, \dots, k_{\max}$. The iterative process provides an approximation to the eigenvector of $B_0^{-1}A$ corresponding to the minimal eigenvalue of $B_0^{-1}A$, i.e., the algebraically smooth vector corresponding to the current solver $B = B_0$. This last vector $\mathbf{w}_1 = \mathbf{w}^{k_{\max}}$ is then used to build a new AMG hierarchy using Algorithm 2. This AMG hierarchy defines a new operator B_1, which, together with the previous one, B_0 is composed as in Equation (12) and tested on the homogeneous system. The whole process is then iterated. That is, at step $r \geq 1$, we have already built AMG operators $B_0,\ B_1,\ \dots,\ B_{r-1}$. Then, we use them in a multiplicative fashion to iterate on the homogeneous equation $A\mathbf{w} = \mathbf{0}$ starting with a random initial vector. We compute successive iterates $\mathbf{w}^k$, for a fixed number of iterations $k = 1, \dots,\ k_{\max}$, as follows:

$$\mathbf{w}^k = (I - B_{r-1}^{-1}A) \dots (I - B_1^{-1}A)(I - B_0^{-1}A)\mathbf{w}^{k-1}.$$

We note that $\mathbf{w}^k \mapsto \mathbf{0}$ as $k \mapsto \infty$, and since $\mathbf{w} = \mathbf{0}$ is the exact solution, $\mathbf{w}^k$ are the true errors. We monitor their norm $\|\mathbf{w}^k\|_A$ to test the convergence property of the composite AMG operator B defined from:

$$I - B^{-1}A = (I - B_{r-1}^{-1}A) \dots (I - B_1^{-1}A)(I - B_0^{-1}A).$$

If the ratio $\frac{\|\mathbf{w}^{k_{\max}}\|_A}{\|\mathbf{w}^{k_{\max}-1}\|_A}$ stays above $\varrho_{desired}$, we choose the last iterate $\mathbf{w}_r = \mathbf{w}^{k_{\max}}$ as the *algebraically smooth* component and use it to build a new AMG operator B_r and augment the composite operator B with one more component, the operator B_r. To define B_r, we use $\mathbf{w}_r$ in Algorithm 2. The iterated bootstrap process described above is stopped when the composite AMG solver B represented by Equation (12) reaches a desired convergence rate or when a maximum number of hierarchies is built.

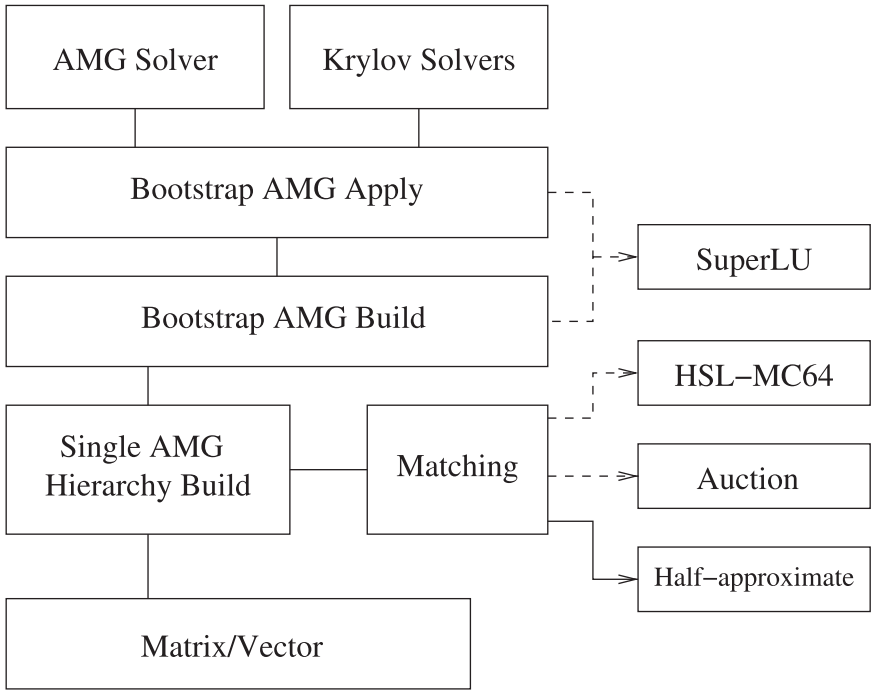

Fig. 1. BootCMatch Software Structure.

If the goal is to use the final B as a preconditioner in the Conjugate Gradient Krylov method, we use its symmetrized version, B^{sym}, defined from the symmetrized product error propagation matrix (assuming the individual AMG operators B_r are constructed linear and symmetric),

$$I - (B^{sym})^{-1} A = (I - B_0^{-1}A)(I - B_1^{-1}A) \dots (I - B_\ell^{-1}A)(I - B_\ell^{-1}A) \dots (I - B_1^{-1}A)(I - B_0^{-1}A).$$

6 BOOTCMATCH SOFTWARE FRAMEWORK

The method described in the previous sections has been implemented in a C-language software framework called *BootCMatch: Bootstrap amg based on Compatible Weighted Matching*. We designed a multi-layer modular framework (see Figure 1) including all the functionalities for building and applying a single hierarchy AMG or a composite AMG relying on the bootstrap process described in Section 5. Each AMG is built on the base of Algorithm 2; the final simple or composite AMG can be applied either as a stand-alone solver or as a preconditioner within a Krylov iterative method. The BootCMatch software code (rel. 0.9), including some testing codes and data sets, is available at the URL in Reference [7]. In particular, at the base layer, we implemented functionalities for basic linear algebra operations on vectors and sparse matrices.[1] The main data structures are those for sparse matrices represented in the Compressed Sparse Row (CSR) storage scheme (`bcm_CSRMatrix`), and for vectors (`bcm_Vector`), respectively. At this level, we also implemented the functionalities for sparse matrix and vector management and for applying basic point-wise relaxation schemes (weighted Jacobi, forward/backward Gauss-Seidel, and symmetrized Gauss-Seidel).

[1] Many of the routines for basic linear algebra are based on the design of those implemented in *hypre rel. 2.9* [23].

The functionalities needed for building each AMG hierarchy are implemented at a medium-level layer, where the main data structure is the general multilevel hierarchy represented in terms of an array of sparse matrices (`bcm_AMGHierarchy`). This structure includes the current-level matrix and the corresponding diagonal, lower triangular, and upper triangular parts, to be used in the pre/post smoothing steps of a general cycle. At the same medium level layer, we implemented the coarsening algorithm described in Algorithm 2; the algorithm makes use of either the half-approximate matching, the HSL-MC64 software, or the simplified auction-type algorithm implemented in the SPRAL software, if available and configured at compile time. Both HSL-MC64 and SPRAL are implemented in Fortran 95, with C interfaces available through the C interoperability layer defined in the Fortran language standard [35]. At the coarsest level of the multigrid hierarchy, we have the option of applying the SuperLU sparse direct solver [33], if configured at build time. At the coarsest level, the multilevel hierarchy, therefore, includes the sparse LU factorization of the coarsest matrix, computed by SuperLU during the set-up phase of each AMG hierarchy. The user defines the algorithmic parameters in the coarsening scheme, in particular the number of pairwise aggregation steps that needs to be applied to define a possibly more aggressive coarsening than the basic pairwise scheme, and also, one can choose to apply one sweep of the weighted Jacobi smoother to the piecewise constant prolongator P_c in Equation (9), in order to define a smoothed aggregation (SA)-type method.

Above this AMG level, we have the functionalities implementing the bootstrap algorithm; these are used to set up the composite solver with a prescribed convergence rate. The composite solver is represented in terms of an array of multilevel hierarchies (`bcm_BootAMG`), which can be applied in a more general way than in Equation (12), including additive and symmetrized-multiplicative composition. The `bcm_BootAMG` hierarchy also includes the smooth vectors computed at each new application of the current composite AMG built during the bootstrap process. In this software layer, we also provide functionalities to apply each multilevel hierarchy in a general $\mu-$fold cycle [47], including as special cases the standard V- and W-cycles. As an alternative, we provide an implementation of a K-cycle [39] with two inner iterations of Flexible Conjugate Gradient (FCG) at each level but the coarsest one (see Algorithm 3.2 in Reference [38]), to improve convergence behavior in the case of unsmoothed prolongators.

Finally, a linear solver layer implements both the use of the composite solver as a stand-alone iterative method, as well as a preconditioned FCG employing the general (single component or composite) AMG as preconditioner. More precisely, we use the version of FCG(1) described in Reference [39], which is equivalent to the standard conjugate gradient method when a SPD preconditioner is used, while it allows to enhance the stability of the standard method when a variable preconditioner is employed.

7 RESULTS AND COMPARISONS

We tested BootCMatch on several sparse SPD matrices arising from different model problems and on some matrices available on the SuiteSparse Matrix Collection [20]. The runs have been carried out on one core of a 2.6GHz Intel Xeon E5-2670, running the Linux 2.6 kernel with the GNU compiler version 4.9, release 5.0 of SuperLU, release 2.3.1 of the C interface to HSL-MC64 and the latest available release of the SPRAL.

We always used the AMG method as preconditioner for the FCG Krylov method. We present both results related to the use of a single component AMG, as well as results obtained with the bootstrap process of Section 5 applied to obtain a composite AMG solver with a prescribed convergence rate.

In all cases except where it is specified otherwise, we solved the linear systems with the right-hand side set equal to the unit vector. The iterations were stopped when the Euclidean norm

of the relative residual reached the tolerance $rtol = 10^{-6}$ or a maximum number of iterations $itmax = 1000$ was reached. We always considered AMG hierarchies built by Algorithm 2, with the maximum size of the coarsest matrix fixed to $maxcoarseset * n^{1/3}$, where n is the matrix dimension so that the cost of the direct solution of the coarsest system is no larger than the cost of a matrix-vector product involving the original matrix; we generally set $maxcoarseset = 40$, while in the case of slow coarsening (i.e., when coarsening ratio of the basic pairwise aggregation scheme is less than 1.2 out of 2 for the current level), we dynamically set $maxcoarseset = 400$. A maximum number of levels was also fixed to 40. In order to have a maximum coarsening ratio of 4, we compose sets of prolongator operators computed by basic Algorithm 2, as described in Section 3, to obtain double pairwise aggregates. Unless otherwise specified, we apply unsmoothed aggregation, since it generally produces the best results on our test cases when coupled with the symmetric K-cycle. In all cases, one sweep of forward/backward Gauss-Seidel relaxation was applied as pre/post-smoother at the finest level, while the SuperLU sparse factorization was applied at the coarsest level. We report the following parameters, which are commonly used to characterize the quality of the convergence and efficiency of the method:

— $nboot$: number of components built for obtaining the desired convergence rate; it is reported only in the case of the application of the bootstrap process for building a composite AMG;
— ρ: estimated convergence rate of the composite AMG; only in the case of composite AMG;
— average number of levels of the AMG components:

$$nlev = \frac{1}{nboot} \sum_{j=1}^{nboot} nlev_j,$$

where $nlev_j$ is the number of levels of the AMG component built at the bootstrap step j;
— average operator complexity of the AMG components defined as:

$$cmplx = \frac{1}{nboot} \sum_{j=1}^{nboot} \frac{\sum_{k=0}^{nlev_j-1} nnz(A_j^k)}{nnz(A^0)},$$

where A_j^k is the matrix at level k at bootstrap step j ($k = 0$ corresponds to the fine level) and $nnz(A_j^k)$ is the number of non-zeros of A_j^k;
— average coarsening ratio of the AMG components built by the bootstrap process, defined as:

$$cratio = \frac{1}{nboot} \sum_{j=1}^{nboot} \frac{1}{nlev_j} \sum_{k=1}^{nlev_j} \frac{n(A_j^{k-1})}{n(A_j^k)},$$

where $n(A_j^k)$ is the size of matrix A_j^k;
— $tbuild$: the execution time in seconds, needed to set up the preconditioner.

When using a single component AMG, i.e., when the bootstrap process is not applied, we have $nboot = 1$, the above parameters correspond to definitions commonly used in (V-cycle) AMG. Finally, we report the number of iterations and execution times of the preconditioned FCG as it and $tsolve$, respectively.

7.1 Test Cases

Our first model problem is the following anisotropic two-dimensional PDE on the unit square, with homogeneous Dirichlet boundary conditions imposed:

$$-div\ (K\ \nabla u) = f,$$

where K is the coefficient matrix

$$K = \begin{bmatrix} a & c \\ c & b \end{bmatrix}, \quad \text{with} \quad \begin{cases} a = \epsilon + \cos^2(\theta) \\ b = \epsilon + \sin^2(\theta) \\ c = \cos(\theta)\sin(\theta) \end{cases}$$

The parameter $0 < \epsilon \leq 1$ defines the strength of anisotropy in the problem, while the parameter θ specifies the direction of anisotropy. In the following, we discuss results related to $\epsilon = 0.001$ and $\theta = 0, \pi/8, \pi/4$, for a total of three test cases, which we refer to as $Pi_{i=1,2,3}$, respectively. The problem was discretized with the Matlab PDE toolbox, using linear finite elements on triangular meshes of three different sizes (168, 577; 673, 025; 2, 689, 537), obtained by uniform refinement.

A second set of test cases comes from the discretization of Lamé equations for linear elasticity. These are equilibrium equations written in terms of the displacement field $\mathbf{u}$:

$$\mu\Delta\mathbf{u} + (\lambda + \mu)\nabla(div\ \mathbf{u}) = \mathbf{f} \qquad \mathbf{x} \in \Omega,$$

where $\mathbf{u} = \mathbf{u}(\mathbf{x})$ is the displacement vector, Ω is the spatial domain, and λ and μ are the Lamé constants. A mix of Dirichlet boundary conditions and so-called traction conditions are usually applied to have a unique solution. Discretization of the vector equation leads to systems of equations whose coefficient matrix is SPD. The matrix admits two commonly used block forms. If each scalar component of the displacement vector is chosen as block unknown, this leads to the so-called *unknown-based* [42] matrix ordering; i.e., the coefficient matrix admits a block form where each diagonal block corresponds to the matrix coming from the discretization of Laplace equation for each unknown component. In the case of the so-called *node-based* ordering, at each grid point, all scalar components are grouped together, which results in the final matrix to have small block entries of dimension 2×2 or 3×3 (in two or three space dimensions). We considered Lamé equations on a beam characterized by $\mu = 0.42$ and $\lambda = 1.7$; one side of the beam is considered fixed and the opposite end is pushed downward. The problem was discretized using linear finite elements on triangular (2D) and tetrahedral meshes (3D); different mesh sizes were obtained by uniform refinement using the software package MFEM [36]. In the following, we refer to the above test cases as *LE2DU* and *LE3DU* when the unknown-based ordering is applied, and *LE2DN* and *LE3DN* when the node-based ordering is applied. Three different sizes are considered for the 2D cases (66, 690; 264, 450; 1, 053, 186) as well as for the 3D cases (15, 795;111, 843; 839, 619). It is well known that efficient AMG for linear elasticity requires *a priori* knowledge of the smooth errors, i.e., of the so-called rigid body modes (three vectors in 2D and six vectors in 3D problems), and that they are well represented on all coarse levels (see, e.g., Ref. [4]).

For the above test cases, we also considered the corresponding diagonally scaled matrices $D^{-1/2}AD^{1/2}$, where A is the matrix of the original problem and D is its main diagonal so that the scaled matrix has diagonal entries equal to 1. These test cases are referred as *LE2DUS*, *LE3DUS*, *LE2DNS*, and *LE3DNS*, respectively. Rescaling the matrices generally changes the features of the smooth errors components and can lead to a significant deterioration of the convergence rate of standard AMG.

We also include tests on some SPD matrices from the SuiteSparse Matrix Collection; Table 1 summarizes the main features of the selected matrices from that set.

Finally, we present results obtained on linear systems (matrices and right-hand sides) arising from the Alya code, a multiphysics parallel solver developed at the Barcelona Supercomputing Center for atmospheric flow simulations [3]. The code implements a stabilized finite-element formulation based on the variational multiscale method for solving the Reynolds-Averaged Navier-Stokes equations coupled with a $k - \epsilon$ turbulence model in a 3D domain representing the Atmospheric Boundary Layer. Using a fractional step approach, the velocity field is decoupled from the

Table 1. Main Features of Selected Matrices

Matrix Name	Group	Application Field	n	nnz	$cond$
PDE problems					
crystm03	Boeing	material pb.	24,696	583,770	264.003
sts4098	Cannizzo	structural eng.	4,098	72,356	2.17×10^8
qa8fm	Cunningham	acoustics pb.	66,127	1,660,579	109.601
thread	DNVS	structural eng.	29,736	4,444,880	2.57×10^{10}
crankseg_1	GHS_psedef	structural eng.	52,804	10,614,510	2.23×10^8
non PDE problems					
obstclae	GHS_psedef	quadratic obstacle pb.	40,000	197,608	40.997
chem97ztz	Bates	statistical pb.	2,541	7,361	247.219
bundle1	Lourakis	computer graphycs	10,581	770,811	1,004.24
cvxbqp1	GHS_psedef	optimization	50,000	349,968	9.31×10^6

pressure field and two SPD linear systems have to be solved at each time step. The velocity vector field is discretized using a node-based approach, leading to a matrix with 3×3 block entries. Our test case is related to wind flow simulations in a realistic scenario for wind farm applications obtained by the Energy Oriented Center of Excellence (EoCoE) European project. Again, in this test case, we consider three different grid sizes, corresponding to pressure matrices of size 305, 472; 790, 856; and 2, 224, 476; and velocity matrices of size 916, 416; 2, 372, 568; 6, 673, 428. The presence of high numerical anisotropy, due to the use of block-structured non-uniform hexahedral grid, makes this test case very challenging for standard AMG.

7.1.1 Comparison of the Matching Algorithms. We begin our discussion with the results of the application of the three available matching algorithms to the test cases $Pi_{i=1,\ldots,3}$ for increasing mesh size. To this aim, we only consider a single AMG component as preconditioner without using bootstrap. In Table 2, we report the parameters related to set-up costs, while number of iterations and execution times for application of the preconditioner are reported in Table 3. Using the matching algorithm implemented in HSL-MC64 produces hierarchies with the smallest operator complexity and generally the largest coarsening ratio. The operator complexity of the hierarchies, for all test cases and mesh sizes, is about 1.34 when HSL-MC64 is used, about 1.41 in the case of the half-approximate matching, and 1.37 with the auction-type matching.

These operator complexities result from a coarsening ratio, which is more than 3.88 out of 4 for the HSL-MC64 matching when the medium and largest mesh sizes are used. The auction-type matching also produces good coarsening ratio for increasing mesh sizes, resulting in hierarchies with the same number of levels as the previous case, with a small increase in the operator complexities. Looking at the execution times for building the hierarchies, we can observe that HSL-MC64 matching, as expected, requires the largest times: for all mesh sizes, it takes about twice as long as when auction-type matching was used. The good operator complexities obtained by the HSL-MC64 matching have a positive impact in the solution phase, where we can observe that we generally obtain the best execution times per iteration. Both the half-approximate matching and the auction-type matching produce good quality coarse matrices, resulting in a number of iterations to converge, which is smaller than that required by the AMG based on HSL-MC64 matching.

Table 2. BootCMatch: Building a Single Component AMG with Different Matching Algorithms

BootCMatch: Comparison among matching algorithms											
P1				*P2*				*P3*			
nlev	*cmpx*	*cr*	*tbuild*	*nlev*	*cmpx*	*cr*	*tbuild*	*nlev*	*cmpx*	*cr*	*tbuild*
Half-approximate Matching											
5	1.40	3.41	0.46	5	1.40	3.41	0.46	5	1.40	3.42	0.46
6	1.41	3.10	1.90	6	1.41	3.10	1.90	6	1.41	3.10	1.88
7	1.41	3.15	7.81	7	1.41	3.14	7.82	7	1.41	3.14	7.81
Auction-type Matching											
5	1.37	3.27	0.49	5	1.37	3.29	0.49	5	1.37	3.30	0.48
5	1.37	3.73	2.15	5	1.37	3.74	2.14	5	1.37	3.74	2.08
6	1.37	3.74	9.34	6	1.37	3.73	9.23	6	1.37	3.74	9.11
HSL-MC64 Matching											
5	1.34	3.42	0.91	5	1.34	3.41	0.96	5	1.34	3.42	0.94
5	1.34	3.90	3.86	5	1.34	3.90	4.17	5	1.34	3.89	4.22
6	1.34	3.89	17.22	6	1.34	3.89	19.71	6	1.34	3.88	19.91

Table 3. BootCMatch: Applying a Single Component AMG, Built with Different Matching Algorithms, as Preconditioner for FCG

BootCMatch: Comparison among matching algorithms					
P1		*P2*		*P3*	
it	*tsolve*	*it*	*tsolve*	*it*	*tsolve*
Half-approximate Matching					
81	2.94	82	3.01	83	3.01
105	17.28	103	16.97	107	17.48
127	91.27	128	91.99	132	95.01
Auction-type Matching					
81	2.71	83	2.74	85	2.77
106	15.39	108	15.52	110	15.71
132	84.29	133	84.72	136	85.93
HSL-MC64 Matching					
86	2.64	91	2.80	86	2.63
114	14.98	116	15.29	119	15.66
146	83.76	143	82.43	155	89.60

The good behavior of the AMG based on near-optimal matching confirms that the corresponding vector space $\mathcal{R}ange(P_f)$, which can be explicitly computed at each level of Algorithm 2 by using the formula in Equation (11), is handled well by the AMG smoother. As discussed in Section 2.2, a measure of the effectiveness of the smoother on the space $\mathcal{R}ange(P_f)$ can be obtained by estimating the convergence rate of the related compatible relaxation. In Table 4, we summarize an estimate

Table 4. Compatible Relaxation Convergence Rates at Each Level of Algorithm 2

	Test case *P2* corresponding to the smallest size		
lev	Half-approximate Matching	Auction-type Matching	HSL-MC64 Matching
1	0.897	0.831	0.849
2	0.815	0.806	0.832
3	0.812	0.773	0.811
4	0.782	0.736	0.762
5	0.750	0.706	0.722
6	0.735	0.686	0.696
7	0.714	0.676	0.685

Table 5. BootCMatch: Single Component AMG, Built with Auction-type Matching, as Preconditioner for FCG

BootCMatch: single component AMG building and application						
	Building				*Application*	
name	*nlev*	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
PDE problems						
crystm03	4	1.73	3.32	0.25	8	0.05
sts4098	3	1.45	2.78	0.03	102	0.08
qa8fm	5	1.90	2.94	0.86	8	0.17
thread	4	1.24	3.17	1.86	1,000	29.87
crankseg_1	4	1.24	3.67	5.40	160	10.65
non-PDE problems						
obstclae	4	1.36	3.30	0.05	7	0.03
chem97ztz	2	1.95	1.05	0.003	4	0.001
bundle1	3	1.61	2.09	1.09	13	0.07
cvxbqp1	4	1.52	3.50	0.19	1,000	4.74

of the compatible relaxation convergence rate $\rho_f = \|I - M_f^{-1}A_f\|_{A_f}$ obtained by computing the ratio r_k/r_{k-1}, where r_k is the residual at the iteration k, and M_f is a weighted Jacobi smoother (with weight equal to 1/3) applied to A_f for a sufficiently large number of iterations. In these experiments, we set $k = 20$ and, for sake of space, we report results for the test case *P2* with the smallest size. We can observe that, at each level *lev* of the constructed hierarchy, the compatible relaxation convergence rate measured for the auction-type matching is generally better than that obtained by the other two algorithms. Furthermore, the auction-type matching produces the best total (set-up and solve) execution times. Since similar results are generally obtained for all our test cases, in the following, we report results obtained using the auction-type matching.

7.2 Further Results with Single Component AMG

In Table 5, we summarize results obtained when a single AMG component built by BootCmatch with the auction-type matching is applied as a preconditioner for FCG to the matrices arising from the SuiteSparse Matrix Collection. We observe that in all cases except *thread* and *cvxbqp*1, BootCmatch is able to build, in an efficient way, a good preconditioner with reasonable coarsening ratio and operator complexity. In the case of the *chem*97*ztz* matrix, Algorithm 2 is unable to effectively

Table 6. BootCMatch: Single Component AMG, Built with Auction-type Matching, as Preconditioner for FCG

BootCMatch: single component AMG building and application					
Building				*Application*	
nlev	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
Test Alya—Pressure					
5	1.30	3.81	2.01	97	8.82
5	1.31	3.83	5.46	86	21.89
6	1.30	3.77	15.08	79	58.08
Test Alya—Velocity					
6	1.33	3.35	28.05	69	47.67
6	1.32	3.71	90.22	46	101.78
7	1.31	3.43	264	125	838

coarsen the matrix, leading to a two-level preconditioner with a high operator complexity of 1.95, which corresponds to a coarsening ratio of only 1.05 out of expected value of 4, and a coarsest matrix size of 2, 410. For the *cvxbqp*1 matrix, the convergence of the preconditioned FCG appears very slow: to reach the prescribed accuracy, 1,789 iterations were needed. For the *thread* matrix, the solver stalled around a relative residual of about 10^0 after $1,000$ iterations. However, we observed that if we apply BootCMatch for building a bootstrap AMG based on a symmetrized multiplicative composition, a composite preconditioner with seven components and an estimated convergence rate of 0.92, with a building cost of 40.88 seconds, we can attain convergence in $1,928$ iterations within 877.58 seconds of solver time.

Results obtained when a single component AMG was used as preconditioner of the FCG solver on the systems arising from the Alya code are summarized in Table 6. For all mesh sizes, and for both scalar and vector systems, BootCMatch is able to obtain an efficient preconditioner with a small operator complexity of no more than 1.33; in all the cases, the coarsening based on matching produces sufficiently large coarsening ratio and effective AMG hierarchies. In the case of the finest mesh size, corresponding to a pressure matrix with more than 58 millions non-zeros and a velocity matrix with more than 530 millions non-zeros, the cost for building and applying a single component AMG is about 10^{-6} seconds per non-zero entry, for both the pressure and velocity system.

7.3 Applying Bootstrap AMG Preconditioner

We now turn to the application of a bootstrap AMG when a convergence ratio $\rho = 0.8$ is prescribed; we consider the test cases arising from linear elasticity, where, as expected, single component AMG built on the base of a single smooth vector is not effective.

To set up the bootstrap AMG, we choose symmetrized multiplicative composition of the multiple hierarchy built within the iterative process. The composite AMG is applied as a solver to the homogeneous system associated to the original coefficient matrix for a number of iterations $\nu = 15$, in order to estimate its asymptotic convergence rate. In Table 7, we report results obtained when applying a double pairwise unsmoothed aggregation for each hierarchy and a K-cycle for the application, as in the results discussed in the previous sections. In all cases, the bootstrap AMG is able to obtain an asymptotic convergence rate smaller than the prescribed one, leading to a small number of iterations of the preconditioned FCG in the solution phase; this, in turn, means having

Table 7. BootCMatch: FCG Preconditioned with a Bootstrap AMG of Desired Convergence Rate, When Double Pairwise Unsmoothed Aggregation and K-cycle Are Applied

BootCMatch: Bootstrap AMG building and application							
Building						*Application*	
nboot	ρ	*nlev*	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
LE2DU							
5	0.80	4	1.41	3.75	6.24	16	1.90
6	0.66	5	1.43	3.38	41.69	15	10.85
7	0.59	6	1.43	3.41	254.27	14	54.99
LE3DU							
5	0.64	4	1.68	3.05	2.09	12	0.48
4	0.68	5	1.72	3.19	13.35	16	4.75
9	0.51	6	1.32	3.25	1,483.11	11	178.07
LE2DN							
6	0.57	4	1.41	3.75	8.49	12	1.75
6	0.67	5	1.43	3.37	40.54	15	10.65
7	0.76	6	1.43	3.40	238.56	18	65.48
LE3DN							
7	0.50	4	1.31	3.04	9.60	9	1.13
7	0.65	5	1.32	3.18	88.81	13	14.42
8	0.76	6	1.32	3.24	1,035.68	13	151.20

a final solver with good scalability with respect to the problem size. As expected, the number of the AMG components to be built for a prescribed accuracy generally increases with the matrix size for both 2D and 3D problems; however, no more than nine components are needed for the largest size LE3DU test case, where a final composite AMG with a very small asymptotic convergence rate of 0.51 is obtained. The unknown ordering used for discretization does not seem to have a large impact on the coarsening process, especially in the 2D cases; indeed, the average number of levels, the average operator complexity and the average coarsening ratio of all the AMG components have similar values for both orderings.

In the 3D cases, the impact of the ordering appears to be more important, although it also depends on the matrix size. However, we observe that the use of bootstrap AMG, with no *a priori* information on rigid body modes, improves the convergence rate of the final preconditioner and the large set-up costs are justified by small solution times.

A substantial fraction of the set-up cost is due to the procedure for estimating the asymptotic convergence rate and the generation of the new approximate smooth vector; hence, it can be significantly reduced if we are able to fix, *a priori*, the number of components to be built by using *a priori* information on the smooth vector space. Furthermore, in many applications, we need to solve many linear systems having the same coefficient matrix; therefore, large building costs can be justified if they enable a more efficient solution. Similar results are obtained on the diagonally scaled problems as shown in Table 8. In Table 9, we show results obtained for the diagonally scaled problems using bootstrap AMG with prescribed desired convergence rate, where each AMG component is built by composition of four consecutive basic pairwise prolongators obtained by Algorithm 2. In this way an aggressive coarsening is produced with a maximum coarsening ratio of 16 and the

Table 8. BootCMatch: FCG Preconditioned with a Bootstrap AMG of Desired Convergence Rate, When Double Pairwise Unsmoothed Aggregation and K-cycle Are Applied

BootCMatch: Bootstrap AMG building and application							
Building						*Application*	
nboot	ρ	*nlev*	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
LE2DUS							
6	0.59	4	1.41	3.75	8.42	12	1.72
8	0.63	5	1.43	3.36	68.95	11	10.80
8	0.67	6	1.43	3.40	319.41	15	66.46
LE3DUS							
5	0.78	4	1.67	3.05	2.01	13	0.49
5	0.75	5	1.72	3.20	18.92	15	5.52
9	0.69	6	1.32	3.24	1,478.26	12	192.28
LE2DNS							
6	0.55	4	1.41	3.76	8.37	10	1.46
6	0.79	5	1.43	3.38	40.07	15	10.39
8	0.66	6	1.43	3.41	300.24	13	54.78
LE3DNS							
6	0.73	4	1.31	3.03	7.35	11	1.15
8	0.48	5	1.32	3.19	108.49	10	12.79
9	0.76	6	1.32	3.25	1,243.31	12	156.12

final prolongators are computed by applying one sweep of a weighted-Jacobi smoother, as discussed in Section 3, considering a more accurate smoothed aggregation-type approach. A simple V-cycle is employed for the application of each hierarchy. The average of the operator complexities of the AMG hierarchies remains generally much smaller than two in all cases, except the first two matrix sizes of the LE3DUS test. More components are generally needed, with respect to the results in Table 8, to reach the prescribed asymptotic convergence; however, due to the general reduction in the number of levels of each hierarchy and in their operator complexities, the set-up times are reduced for the 2D test cases, whereas in the 3D cases, building costs have a moderate increase. We note that, even in such a case, the largest fraction of the set-up times corresponds to the estimation of the asymptotic convergence rate and the generation of the new approximate smooth vector. Nevertheless, the scalability of the final solver remains fairly good, and the timings for solving the systems are reduced in all test cases.

7.4 Comparisons with Related Works Based on Unsmoothed Aggregation

In this section, we report, for comparison purposes, some results obtained by using the academic version of AGMG (rel. 3.2) software code [38]. Our aim is to compare the behavior of single component AMG obtained by BootCMatch with a similar AMG operator based on unsmoothed aggregation. We run AGMG with the default setting for SPD matrices, i.e., each hierarchy is built by the double pairwise unsmoothed aggregation scheme described in [37, 38], where matching among the unknowns is driven by a usual measure of strength of connections, and a K-cycle using two inner iterations is applied as preconditioner in the FCG Krylov solver. The sparse LU factorization

Table 9. BootCMatch: FCG Preconditioned with a Bootstrap AMG of Desired Convergence Rate, When Aggressive Coarsening with Smoothed Aggregation and V-cycle Are Applied

BootCMatch: Bootstrap AMG building and application							
Building						*Application*	
nboot	ρ	*nlev*	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
LE2DUS							
7	0.53	3	1.30	9.08	8.47	10	1.06
7	0.77	3	1.20	11.26	36.70	15	6.66
9	0.64	4	1.23	10.18	248.51	15	37.10
LE3DUS							
5	0.73	3	2.68	7.63	2.86	15	0.46
8	0.72	3	2.26	10.27	42.69	12	4.70
9	0.78	4	1.37	9.62	1,498.45	13	140.68
LE2DNS							
7	0.60	3	1.30	9.08	8.19	11	1.12
8	0.58	3	1.21	11.22	43.71	13	6.45
9	0.67	4	1.23	10.17	228.60	16	36.60
LE3DNS							
7	0.57	3	1.41	7.49	10.97	10	1.01
8	0.68	3	1.29	10.07	108.39	12	10.49
9	0.70	4	1.37	9.62	1,289.79	14	119.94

provided in MUMPS [2] is used as the coarsest solver. In Table 10, we report results obtained on the test cases $Pi_{i=1,2,3}$. If we compare the above results with the ones reported in Table 2, corresponding to the auction-type matching and the best BootCMatch execution times, we can observe that AGMG generally builds hierarchies with one less level than those obtained by BootCMatch, and therefore, with slightly smaller operator complexities, even if BootCMatch has an average coarsening ratio that is generally better than AGMG. This is due to the different coarsening strategy employed by the two codes. AGMG obtains smaller execution times both for building and for applying the solver; this is essentially related to the way AGMG computes the coarse matrices. The prolongators involved in the AGMG unsmoothed aggregation scheme are simple 0-1 matrices whose action can be implemented by simple gather/scatter operations without forming the matrices explicitly; this is in contrast with the general form of the prolongators/restrictions defined by BootCMatch, which require full-fledged matrix-vector products. In Table 11, we report results obtained by AGMG on the test cases arising from the SuiteSparse Matrix Collection. We can observe that AGMG, in all test cases except *obstclae* and *chem*97*ztz*, is unable to coarsen the matrices in an efficient way, showing a very small coarsening ratio. Indeed, for the *crystm*03 and $qa8fm$ matrices, we see a coarsening ratio exactly 1, i.e., the matrices are not coarsened at all, and reverting to a one-level preconditioner means we are effectively solving the original system with a sparse LU factorization. In general, the AGMG coarsening algorithm results in a large operator complexity, and set-up and solve times that are larger than those achieved by BootCMatch. In the case of *obstclae* and *chem*97*ztz*, the average coarsening ratio of AGMG appears better than that obtained by BootCMatch, producing smaller matrix dimensions at corresponding levels in the

Table 10. AGMG for $Pi_{i=1,2,3}$ Test Cases

AGMG building and application					
Building				*Application*	
nlev	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
*P*1					
4	1.36	3.14	0.11	91	1.84
5	1.35	3.25	0.38	121	9.14
5	1.35	3.27	1.47	155	51.7
*P*2					
4	1.36	3.13	0.09	91	1.54
5	1.35	3.23	0.32	116	7.41
5	1.35	3.22	1.35	148	44.2
*P*3					
4	1.36	3.17	0.1	93	2.09
5	1.36	3.29	0.48	121	11.4
5	1.35	3.28	1.34	152	44.9

Table 11. AGMG for Test Cases from the SuiteSparse Matrix Collection

AGMG building and application						
	Building				*Application*	
name	*nlev*	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
PDE problems						
crystm03	2	2.0	1.0	3.83	1	0.03
sts4098	2	1.74	1.34	0.06	38	0.1
qa8fm	2	2.0	1.0	16.9	1	0.34
thread	3	1.85	1.69	3.22	1,000 (NC)	84.2
crankseg_1	5	2.52	1.63	3.99	79	21.4
non PDE problems						
obstclae	4	1.32	4.08	0.04	9	0.05
chem97ztz	2	1.09	3.71	0.001	8	0.002
bundle1	2	1.61	1.75	0.42	13	0.08
cvxbqp1	3	3.0	1.0	1.77	2	0.13

hierarchy; therefore, the final solver generally requires more iterations than BootCMatch, leading to a slightly larger solve time.

Finally, in Table 12, we report results obtained by AGMG on the test cases arising from the Alya simulation code. We can observe that in all the cases, BootCMatch outperforms AGMG both in the setup and in the application of the preconditioner. In particular, AGMG exhibits large difficulties in the coarsening of the vector systems, where very small coarsening ratios are observed. We also notice that in the case of the smallest size velocity system, AGMG is not convergent, producing an increase of the relative residual in the solution process.

Table 12. AGMG for Alya Test Cases

AGMG building and application					
Building				*Application*	
nlev	*cmpx*	*cr*	*tbuild*	*it*	*tsolve*
Test Alya—Pressure					
5	1.39	2.7	0.56	100	8.3
7	1.50	2.6	1.83	92	23.8
7	1.52	2.9	3.78	96	74.0
Test Alya—Velocity					
5	1.9	1.7	313	1,000	2,890
6	1.78	1.7	73.1	121	444
6	1.65	1.9	547	203	14,200

8 CONCLUSIONS

In this article, we described some new features of an adaptive AMG method aimed at handling general SPD linear systems, without exploiting any *a priori* information about the problem at hand. We also presented the software framework *BootCMatch*, which implements all the functionalities for building and applying the method as stand-alone solver or as preconditioner in a Krylov method. Results on a variety of test cases show that the method works in a fairly reliable and efficient way. Work in progress aims at reducing the computational costs of the solve phase of the adaptive AMG, by using already computed hierarchies and multiple smooth vectors to generate a more effective AMG hierarchy, thus reducing the operator complexity of the final AMG. Future work includes the extension of the method and the related software to unsymmetric problems as well as effective parallel implementation on modern architectures. Preliminary results related to a decoupled parallel implementation of the coarsening procedure described in Algorithm 2 can be found in Reference [1].

ACKNOWLEDGMENTS

We wish to thank Herbert Owen (BSC, Barcelona) for making available the test problems arising from the Alya simulation code. We also thank the anonymous reviewers for their comments, which helped us to obtain an improved version of the article.

REFERENCES

[1] A. Abdullahi, P. D'Ambra, D. di Serafino, and S. Filippone. 2018. Parallel aggregation based on compatible weighted matching for AMG. In *Large Scale Scientific Computing. LSSC 2017. (Lecture Notes in Computer Science)*, I. Lirkov and S. Margenov (Eds.), Vol. 10665. Springer, Cham, Switzerland, 563–571.

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J. Y. L'Excellent. 2001. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.* 23 (2001), 15–41.

[3] M. Avila, A. Folch, G. Houzeaux, B. Eguzkitza, L. Prieto, and D. Cabezón. 2013. A parallel CFD model for wind farms. *Procedia Computer Science* 18 (2013), 2157–2166. International Conference on Computational Science, 2013.

[4] A. H. Baker, Tz. V. Kolev, and U. M. Yang. 2010. Improving algebraic multigrid interpolation operators for linear elasticity. *Numer. Linear Algebra Appl.* 17 (2010), 495–517.

[5] D. P. Bertsekas. 1988. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research* 14 (1988), 105–123.

[6] D. P. Bertsekas. 1992. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications* 1 (1992), 7–66.

[7] BootCMatch. 2017. Bootstrap algebraic multigrid based on compatible weighted matching. Retrieved from https://github.com/bootcmatch/BootCMatch.

[8] A. Brandt. 2000. General highly accurate algebraic coarsening. *Electronic Transactions on Numerical Analysis* 10 (2000), 1–20.
[9] A. Brandt, J. Brannick, K. Kahl, and I. Livshits. 2015. Algebraic distance for anisotropic diffusion problems: Multilevel results. *Electronic Transactions on Numerical Analysis* 44 (2015), 472–496.
[10] A. Brandt, J. Brannick, K. Kahl, and I. Livshits. 2015. Bootstrap algebraic multigrid: Status report, open problems, and outlook. *Numerical Mathematics: Theory, Methods and Applications* 8 (2015), 112–135.
[11] A. Brandt, J. Brannick, K. Kahl, and I. Livshitz. 2011. Bootstrap AMG. *SIAM J. Sci. Comput.* 33 (2011), 612–632.
[12] A. Brandt, S. McCormick, and J. Ruge. 1985. Algebraic multigrid (AMG) for sparse matrix equations. In *Sparsity and its Applications*. Cambridge Univ. Press, Cambridge, 257–284.
[13] J. Brannick, Y. Chen, J. Kraus, and L. Zikatanov. 2013. Algebraic multilevel preconditioners for the graph Laplacian based on matching in graphs. *SIAM J. Numer. Anal.* 51 (2013), 1805–1827.
[14] J. Brannick, Y. Chen, and L. Zikatanov. 2012. An algebraic multilevel method for anisotropic elliptic equations based on subgraph matching. *Numer. Linear Algebra Appl.* 19 (2012), 279–295.
[15] J. Brannick and R. D. Falgout. 2010. Compatible relaxation and coarsening in algebraic multigrid. *SIAM J. Sci. Comput.* 32 (2010), 1393–1416.
[16] M. Brezina, R. D. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. 2005. Adaptive smoothed aggregation αSA multigrid. *SIAM Rev.* 47 (2005), 317–346.
[17] M. Brezina, R. D. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. 2006. Adaptive algebraic multigrid. *SIAM J. Sci. Comput.* 27 (2006), 1261–1286.
[18] P. D'Ambra and P. S. Vassilevski. 2013. Adaptive AMG with coarsening based on compatible weighted matching. *Comput. Visual Sci.* 16 (2013), 59–76.
[19] P. D'Ambra and P. S. Vassilevski. 2016. Adaptive AMG based on weighted matching for systems of elliptic PDEs arising from displacement and mixed methods. In *Progress in Industrial Mathematics at ECMI 2014 (Mathematics in Industry)*, Russo G. et al. (Eds.), Vol. 22. Springer-Verlag, Berlin, Germany, 1013–1020.
[20] T. A. Davis and Y. Hu. 2011. The University of Florida sparse matrix collection. *ACM Trans. Math. Software* 38, 1 (2011), 1:1–1:25.
[21] R. Diestel. 2010. *Graph Theory, 4th ed.*Springer, Heidelberg, GTM 173.
[22] I. S. Duff and J. Koster. 2001. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.* 22 (2001), 973–996.
[23] I. S. Duff and S. Pralet. 2005. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Anal. Appl.* 27 (2005), 313–340.
[24] R. D. Falgout, J. E. Jones, and U. Meyer Yang. 2006. The design and implementation of Hypre, a library of parallel high-performance preconditioners. In *Numerical Solutions of Partial Differential Equations on Parallel Computers (Lecture Notes in Computational Science and Engineering)*, A. M. Bruaset and A. Tveito (Eds.), Vol. 15. Springer-Verlag, Berlin, Germany, 267–294.
[25] R. D. Falgout and P. S. Vassilevski. 2004. On generalizing the algebraic multigrid framework. *SIAM J. Numer. Anal.* 42 (2004), 1669–1693.
[26] R. D. Falgout, P. S. Vassilevski, and L. T. Zikatanov. 2005. On two-grid convergence estimates. *Numer. Linear Algebra Appl.* 12 (2005), 471–494.
[27] M. Hagemann and O. Schenk. 2006. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM J. Sci. Comput.* 28 (2006), 403–420.
[28] M. Halappanavar, J. Feo, O. Villa, A. Tumeo, and A. Pothen. 2012. Approximate weighted matching on emerging manycore and multithreaded architectures. *Int. J. High Perform. Comput. Appl.* 26 (2012), 413–430.
[29] J. Hogg. 2016. Sparse Parallel Robust Algorithms Library (SPRAL). Retrieved from https://github.com/ralna/spral.
[30] J. Hogg and J. Scott. 2013. Optimal weighted matchings for rank-deficient sparse matrices. *SIAM J. Matrix Anal. Appl.* 34 (2013), 1431–1447.
[31] J. Hogg and J. Scott. 2015. On the use of suboptimal matchings for scaling and ordering sparse symmetric matrices. *Numer. Linear Algebra Appl.* 22 (2015), 648–663.
[32] H. Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2 (1955), 83–97.
[33] X. S. Li. 2005. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Software* 31, 3 (September 2005), 302–325.
[34] O. E. Livne. 2004. Coarsening by compatible relaxation. *Num. Linear Alg. Appl.* 11 (2004), 205–227.
[35] M. Metcalf, J. Reid, and M. Cohen. 2011. *Modern Fortran Explained* (4th ed.). Oxford University Press, Inc., New York, NY.
[36] MFEM. 2015. Modular finite element methods. mfem.org. (2015).
[37] A. Napov and Y. Notay. 2012. An algebraic multigrid method with guaranteed convergence rate. *SIAM J. Sci. Comput.* 34 (2012), A1079–A1109.

[38] Y. Notay. 2010. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis* 37 (2010), 123–146.
[39] Y. Notay and P. S. Vassilevski. 2008. Recursive Krylov-based multigrid cycles. *Numer. Linear Algebra Appl.* 15 (2008), 473–487.
[40] A. Pothen, F. Dobrian, and M. Halappanavar. 2013. Matchbox, a library of graph matching algorithms. (2013). http://www.cs.odu.edu/ mhalappa/matching/.
[41] R. Preis. 1999. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS'99 (Lecture Notes in Computer Science)*, J. Dongarra, K. Madsen, and J. Wasniewski (Eds.), Vol. 1563. Springer-Verlag, Berlin, Germany, 259–269.
[42] J. W. Ruge. 1986. AMG for problems of elasticity. *Appl. Math. Comput.* 19 (1986), 293–309.
[43] J. W. Ruge and K. Stüben. 1987. Algebraic multigrid (AMG). In *Multigrid Methods*, S. F. McCormick (Ed.). SIAM, Philadelphia, 73–130.
[44] M. Sathe, O. Schenk, and H. Burkhart. 2012. An auction-based weighted matching implementation on massively parallel architectures. *Parallel Comput.* 38 (2012), 595–614.
[45] The Numerical Analysis Group. 2011. HSL (2011). A collection of Fortran codes for large scale scientific computation. (2011). http://www.hsl.rl.ac.uk.
[46] P. Vaněk, J. Mandel, and M. Brezina. 1996. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems.*Computing* 56, 3 (1996), 179–196.
[47] P. S. Vassilevski. 2008. *Multilevel Block Factorization Preconditioners: Matrix-based Analysis and Algorithms for Solving Finite Element Equations*. Springer, New York.

Received April 2017; revised February 2018; accepted February 2018