ADAPTIVE SMOOTHED AGGREGATION (αSA)

M. BREZINA*, R. FALGOUT†, S. MACLACHLAN†, T. MANTEUFFEL†, S. MCCORMICK†, AND J. RUGE†

Abstract. Substantial effort has been focused over the last two decades on developing multilevel iterative methods capable of solving the large linear systems encountered in engineering practice. These systems often arise from discretizing partial differential equations over unstructured meshes, and the particular parameters or geometry of the physical problem being discretized may be unavailable to the solver. Algebraic multigrid and multilevel domain decomposition methods of algebraic type have been of particular interest in this context because of their promises of optimal performance without the need for explicit knowledge of the problem geometry. These methods construct a hierarchy of coarse problems based on the linear system itself and on certain assumptions about the smooth components of the error. For smoothed aggregation methods applied to discretizations of elliptic problems, these assumptions typically consist of knowledge of the near-nullspace of the weak form. This paper introduces an extension of the smoothed aggregation method in which good convergence properties are achieved in situations where explicit knowledge of the near-nullspace components is unavailable. This extension is accomplished by using the method itself to determine near-nullspace components and adjusting the coarsening processes accordingly.

Key words. Algebraic Multigrid (AMG), generalized smoothed aggregation, adaptive method.

1. Introduction. Over the last decade, smoothed aggregation (SA; cf. [21, 23, 22, 20, 9]) has emerged as an efficient multilevel algebraic solver for the solution of the algebraic systems obtained by discretizing certain classes of differential equations on unstructured meshes. In particular, SA is often very efficient at solving the systems that arise from problems of 3D thin-body elasticity, a task that can tax traditional algebraic multigrid techniques.

As with classical AMG [4, 18, 19], the standard smoothed aggregation method bases its transfer operators on certain assumptions about the nature of smooth error. For SA applied to discretizations of elliptic partial differential equations, this assumption usually takes the form of explicit knowledge of the near-nullspace of the associated weak form. This knowledge is easy to obtain for large classes of problems. For example, it is simple to determine the near-nullspace for finite element discretizations of second- or fourth-order PDEs, including many nonscalar problems. In more general situations, however, this knowledge may not be readily available. Consider the case where the matrix for a problem is provided without knowledge of how the original problem was discretized or scaled. Seemingly innocuous discretization practices, such as the use of scaled bases, can hamper algebraic multigrid solvers if this scaling is not taken into account. Even the simplest problems discretized on regular grids using standard finite elements can pose serious difficulties if the resulting matrix has been scaled without this information being provided to the solver. Other discretization practices leading to problematic linear systems include the use of exotic bases and systems problems in which different local coordinate systems are used for different parts of the model.

*Department of Applied Mathematics, Campus Box 526, University of Colorado at Boulder, Boulder, CO 80309–0526. *email*: mbrezina@math.cudenver.edu, scott.maclachlan@colorado.edu, tmanteuf@boulder.colorado.edu, stevem@boulder.colorado.edu, and jruge@boulder.colorado.edu. This work was sponsored by the National Institute of Health under grant number 1–R01–EY12291–01, the National Science Foundation under grant number DMS–9706866, and the Department of Energy under grant number DE–FG03–93ER25165.

†Center for Applied Scientific Computation, Lawrence Livermore National Lab, Post Office Box 808, Livermore, CA 94551. *email*: rfalgout@llnl.gov.

To successfully solve such problems when only the matrix is provided, we need a process by which the algebraic multilevel solver can determine how to effectively coarsen the linear system using only information from the system itself. The method we propose here, which we call *adaptive Smoothed Aggregation* (αSA), is an attempt to do just that. αSA is based on the simple principle that applying a linear iterative method to the homogenous problem ($Ax = 0$) reveals error components that the method does not effectively reduce. While this principle is easily stated in loose terms, the resulting algorithm and its implementation can be very subtle. We hope to expose these subtleties in the presentation that follows.

The objective of the setup phase of αSA is therefore to compute a set of vectors, $\mathcal{B}$, that represent error components that relaxation is slow to resolve. Such components are usually referred to by the terms *algebraically smooth*, *near-nullspace*, *near-kernel*, or, in the case of linear elasticity, *rigid body modes*. We simply call them *candidates* here as it is not actually essential that all of the vectors we compute be troublesome components: we use a measure that, in effect, ignores candidates that relaxation efficiently handles. It is also not a problem if we compute redundant or linearly dependent candidates because our approach is designed to select the information we need from the candidate *subspace*. It is, however, important to be certain that the final set of candidates is *rich* in the sense that they combine to represent *all* troublesome components *locally*. The keys in being able to do this are to evolve the multigrid solver by having it compute its own slow-to-converge error components (by way of the homogeneous problem) and to use these new components to properly improve the solver.

The setup phase for αSA is easiest to describe as an adaptive process. We start from a given primitive parent method (possibly a simple relaxation scheme) with error propagation operator M_0, and a current but possibly empty set, $\mathcal{B}$, of candidates (error components that M_0 does not effectively reduce). We attempt to enhance $\mathcal{B}$ by first putting M_0 to the test: given a small number, n, of iterations and a random initial guess, e_0, compute

$$e_n \leftarrow M_0^n e_0. \tag{1.1}$$

If the method performs well in the sense that e_n is much smaller than that of e_0 in an appropriate norm, then it is accepted as the solver and the adaptive scheme stops. Otherwise, the resulting approximation, e_n, is expected to be rich in the error components that are not effectively reduced by M_0, so it is added to the candidate set, $\mathcal{B}$. The new candidates set is then used to construct an improved child method, with error propagation operator M_1. The whole process can then be repeated with M_1 in place of M_0, continuing in this way to generate a sequence of hopefully improving methods, M_k.

Thus, we iterate on the *method* itself, improving the current version by having it compute its own troublesome components–those that it does not effectively reduce–and then adjusting the coarsening process accordingly to produce a new method. Old candidate components are also used in this adjustment process to ensure that the new method continues to reduce them efficiently. This improvement process repeats until the current method shows itself to be capable of efficient solution of the problem of interest. The iteration on the method is called the *adaptive setup phase* (or, simply, the *setup phase*) to distinguish it from the *solver phase* where the resulting method is applied to the target problem. The setup phase is terminated when either the latest incarnation of the method performs satisfactorily or when a prescribed number of steps is reached.

Each new child method is constructed based on components resulting from parent iteration (1.1). The method is modified to reflect the newly computed candidates as soon as they become available. In other words, the method is kept up to date at all times and no more work is done than necessary. In section 3.2, we show how the general setup phase naturally takes the form of a reverse FMG-cycle.

The adaptive strategy outlined above is designed to uncover global error components that a parent method does not handle well. It is crucial to recognize that there are likely to be many such components, so many that, in general, we cannot expect to identify each one individually. Typically, a small but fixed *percentage* of the spectrum of M_k corresponds to troublesome components. Thus, the few candidates that iteration (1.1) identifies must serve as representatives for many smooth components in the coarsening process. This is analogous to the standard SA coarsening processes where the near-kernel is used to represent all smooth components. This representation is accomplished by first taking local segments of each candidate (i.e., by taking the restriction of the candidate to an aggregate and extending it outside the aggregate with zeros) so that the segments sum to the candidate itself. Each segment is then smoothed to enhance the overall approximation property in the sense that it accurately represents similar smooth components. In this way, standard SA constructs a rich set of local representations of the smooth or troublesome components. So too must αSA. Indeed, we need a way to coarsen the system that ensures accurate approximation of the error components that the current candidates represent. Of course, to control storage and CPU costs, we also need to control operator complexity, which involves: limiting the number of candidates that iteration (1.1) produces; exploiting these candidates as fully as we can; and limiting the growth of the number of coarse degrees of freedom.

The smoothed aggregation framework [20] lends itself to this task. It offers fast automatic coarsening with well-understood control over operator complexity due to its typically fixed coarse-operator sparsity pattern. In addition, the process guarantees proper approximation of a given set of functions and their natural localizations during the coarsening process. The resulting coarse-level basis functions are smooth by design and thus suitable for use in a multilevel method. The candidates obtained by iteration (1.1) play the roles of the near-kernel components on which the SA method is based. Thus, in the αSA context, the notion of near-kernel components depends not only on the problem, but also on the current method. In general, however, a troublesome component must have a small Rayleigh quotient, signifying ineffectiveness of relaxation. However, in all but the initial phase (where coarsening has perhaps not yet been constructed) or the final phase (where the method may be efficient), the current candidate must also have a small Rayleigh quotient defined in terms of the current coarse-level projection operator. We do not use this property explicitly in the adaptive process, but keeping it in mind can aid in understanding the development that follows.

Thus, our main goal is to extend applicability of the smoothed aggregation concept to difficult problems for which the original method may perform poorly, possibly due to the lack of explicit knowledge of the near-kernel. The algorithm may also be useful for improving performance in applications that involve multiple right sides, where efforts to improve the method may be amortized over the number of solutions.

In what follows, we develop this modification of the smoothed aggregation method in such a way that good convergence properties are recovered even if explicit knowledge of the near-kernel is either incomplete or lacking altogether. This should facilitate solution in cases where the problem geometry, discretization method, or coefficients of the differential operator are not explicitly known to the solver. At the same time, we strive to keep storage requirements low.

The concept of using a multigrid algorithm to improve itself is not new. Using representative smooth vectors in the coarsening process was first introduced in [15], where interpolation was defined to fit vectors obtained by relaxation of the homogeneous problem. In [4], a variation of this idea was used for recovering typical AMG convergence rates for a badly-scaled scalar elliptic problem. While the method there was very basic and used only one candidate, it contained many of the ingredients of the approach developed below. These concepts were developed further in [16, 17, 19, 14]. The idea of fitting eigenvectors corresponding to the smallest eigenvalues was advocated in [14] and [19], where an AMG algorithm determining these eigenvectors through Rayleigh quotient minimization was outlined. These vectors were, in turn, used to update the AMG interpolation and coarse-grid operators. Most of these ideas were later summarized in [14]. A more sophisticated adaptive framework appropriate for the standard AMG is currently under investigation [7].

Another method of the type developed here is the Bootstrap AMG scheme proposed recently by Brandt [3, 5]. It differs somewhat from ours in that it starts on the fine grid by iterating on a number of different random initial guesses, with interpolation then constructed to approximately fit the resulting vectors in a least-squares sense.

Various other attempts have been made to allow for the solver itself to determine from the discrete problem the information required to successfully solve it, without a priori assumptions on the form of the smooth error. These include the methods of [13, 8, 6, 11, 12]. All these methods, however, need access to the local finite element matrices of the problem so that they can construct the multigrid transfer operators based on the algebraically smooth eigenvectors of the agglomerated stiffness matrices. Although these methods exhibit attractive convergence properties, their need to construct, store, and manipulate the coarse-level element information typically leads to increased storage requirements compared to those of classical AMG or standard SA. The current method aims to achieve the good convergence properties of the element-based methods without the overhead of the element storage.

This paper is organized as follows. In Section 2, we briefly recall the standard smoothed aggregation method and introduce some notation used throughout the remainder of the paper. Readers who are unfamiliar with the fundamental concepts assumed here may first wish to consult basic references on multigrid (e.g., [10]) and smooth aggregation (e.g., [20]). Section 3 motivates and describes possible strategies to extract the information used to construct improved transfer operators based on the method's iterative history. These strategies can be described as adaptive AMG, in which the method ideally evolves until a cross-point at which further improvement (in terms of convergence rate) is offset by the increased cost of each iteration. Section 4 discusses implementation issues and ways of reducing cost and improving accuracy of the setup phase. Finally, Section 5 presents computational examples demonstrating the performance of the smoothed aggregation method based on the adaptive setup concepts.

2. Smoothed Aggregation. We first briefly recall the smoothed aggregation method and introduce some of the notation used later (see [20] for more detail). Assume that A is an SPD matrix of order n_1 resulting from a discretization of an elliptic second- or fourth-order partial differential equation in $\mathbb{R}^{\mathrm{d}}$, where $d \in \{1, 2, 3\}$.

Our aim is to solve $A_1\mathbf{x} = \mathbf{b}_1$, obtained by scaling $A\mathbf{y} = \mathbf{b}$ by its diagonal part, D:

$$A_1 = D^{-1/2}AD^{-1/2}, \qquad \mathbf{b}_1 = D^{-1/2}\mathbf{b}. \tag{2.1}$$

A hierarchy of coarse problems is generated by the Galerkin recurrence:

$$A_{l+1} = (I^l_{l+1})^T A_l I^l_{l+1}, \tag{2.2}$$

where the *prolongator*, I^l_{l+1}, is defined as the product of a given *prolongation smoother*, S_l, and a *tentative prolongator*, P^l_{l+1}:

$$I^l_{l+1} = S_l P^l_{l+1}, \tag{2.3}$$

$l = 1, \ldots, L-1$. Suppose we are given a smoothing procedure for each level $l \in \{1, \ldots, L\}$ system, $A_l\mathbf{x} = \mathbf{b}_l$, of the form

$$\mathbf{x} \leftarrow (I - R_l A_l)\mathbf{x} + R_l\mathbf{b}_l. \tag{2.4}$$

Here, R_l is some simple approximate inverse of A_l (e.g., $R_l = s_l I$, where $s_l \approx \frac{1}{\rho(A_l)}$) for $l = 1, \ldots, L-1$. Assume for simplicity that the coarsest level uses a direct solver: $R_L = A_L^{-1}$. To make use of the existing convergence estimates, we assume that

$$\lambda_{\min}(I - R_l A_l) \geq 0 \quad \text{and} \quad \lambda_{\min}(R_l) \geq \frac{1}{C_R^2 \rho(A_l)},$$

with a constant $C_R > 0$ independent of the level.

The smoothed aggregation iteration can formally be viewed as a standard variational multigrid process with a special choice of transfer operators I^l_{l+1}. One iteration of smoothed aggregation multigrid, $\mathbf{x} \leftarrow \mathbf{AMG}(\mathbf{x}, \mathbf{b_1})$, for solving $A_1\mathbf{x}_1 = \mathbf{b}_1$ is given by setting $\mathbf{AMG} = \mathbf{AMG}_1$, where $\mathbf{AMG}_l(\cdot,\cdot)$, $\quad l = 1, \ldots, L-1$, is defined as follows:

Algorithm 1 ($\mathbf{AMG}_l$).

1. *Pre-smoothing: Apply ν pre-smoothings to $A_l\mathbf{x}_l = \mathbf{b}_l$ of the form*
$\mathbf{x}_l \leftarrow (I - R_l A_l)\mathbf{x}_l + R_l\mathbf{b}_l$.

2. *Coarse-grid correction:*

- **(a)** *Set* $\mathbf{b}_{l+1} = (I^l_{l+1})^T(\mathbf{b}_l - A_l\mathbf{x}_l)$.
- **(b)** *If $l+1 < L$, set $\mathbf{x}_{l+1} = 0$ and solve the coarse grid problem*

$$A_{l+1}\mathbf{x}_{l+1} = \mathbf{b}_{l+1},$$

by γ applications of $\mathbf{x}_{l+1} \leftarrow \mathbf{AMG}_{l+1}(\mathbf{x}_{l+1}, \mathbf{b}_{l+1})$;
else, solve the coarse-level problem directly.

- **(c)** *Correct the solution on level l:* $\mathbf{x}_l \leftarrow \mathbf{x}_l + I^l_{l+1}\mathbf{x}_{l+1}$.

4. *Post-smoothing: Apply ν post-smoothings to $A_l\mathbf{x}_l = \mathbf{b}_l$ of the form*
$\mathbf{x}_l \leftarrow (I - R_l A_l)\mathbf{x}_l + R_l\mathbf{b}_l$.

The components of the method that can be varied to achieve better convergence are the construction of S_l and P^l_{l+1}. A good choice for S_l is described in [20] and scrutinized in more generality in [8]. For our purposes, it suffices to assume that the prolongation smoother, S_l, is the error propagation operator corresponding to

Richardson iteration for the problem on level l, with the particular choice of damping suggested in [20]:

$$S_l = I - \frac{4}{3\ \lambda_l} A_l, \tag{2.5}$$

where $\lambda_l = 9^{1-l}\bar{\lambda}$ and $\bar{\lambda}$ is a bound on the spectral radius of the fine-level matrix: $\rho(A_1) \leq \bar{\lambda}$. With the prolongation smoothers thus chosen, we can concentrate in this paper on the construction of P^l_{l+1}.

NOTE 2.1. Our selection of multigrid smoothing procedure (2.4) and prolongation smoothers S_l follows that of [20], where convergence estimates are obtained. We turn to these results for heuristics in Section 3. □

The construction of operators P^l_{l+1} consists of deriving the sparsity structure and specifying the nonzero values. The nonzero sparsity structure determines the supports of the tentative coarse-grid functions and is specified by way of a decomposition of the set of degrees of freedom associated with operator A_l into an aggregate partition: $\bigcup_{i=1}^{N_{l+1}} \mathcal{A}^l_i = \{1, \ldots, N_l\}$, $\quad \mathcal{A}^l_i \cap \mathcal{A}^l_j = \emptyset$, $\quad 1 \leq i < j \leq N_{l+1}$, $l = 1, \ldots, L-1$, where N_l denotes the number of nodes on level l. Note that the number of aggregates on level l defines the number of nodes on the next level: $N_{l+1} = card(\{\mathcal{A}^l_i\})$. Let n_l denote the number of degrees of freedom on level l and assume at least one degree of freedom is associated with each node, so that $n_l \geq N_l$. Aggregates $\mathcal{A}^l_i$ can be formed based only on the connectivity and strength of connection between the entries of A_l, cf. [23].

Although we illustrate these concepts on the example of a finite element discretization, where the notion of a node should be most familiar to the reader, for us a node is a strictly algebraic entity consisting of a list of degrees of freedom. In fact, the finite element analogy is only possible on the finest level; the degrees of freedom on all other levels have no explicit geometry associated with them. Thus, throughout this paper, a node on level $l + 1 > 1$ is a set of degrees of freedom associated with the coarse basis functions whose discrete supports contain the same aggregate on level l. Thus, each aggregate, $\mathcal{A}$, on level l gives rise to one node on level $l + 1$, and each degree of freedom constituting that node is a coefficient of a particular basis function in the coarse-level basis expansion associated with $\mathcal{A}$.

The second ingredient in constructing generalized aggregation tentative prolongators P^l_{l+1} consists of endowing the sparsity structure derived from the nodal aggregation with appropriate values. Starting with a given matrix, B^1, whose columns represent the near-kernel of the fine-level operator, we construct the tentative prolongators and the coarse-level representation of the near-kernel components simultaneously to satisfy

$$P^l_{l+1} B^{l+1} = B^l, \quad (P^l_{l+1})^T P^l_{l+1} = I. \tag{2.6}$$

This construction of P^l_{l+1} and B^{l+1} is practical and parallelizable because it is achieved by assigning each nodal aggregate a set of columns of P^l_{l+1} with a sparsity structure that is disjoint from all other columns. Thus, obtaining (2.6) amounts to solving a set of local independent orthonormalization problems in which the basis given by the fine-level near-kernel matrix, B^l, restricted to the degrees of freedom of an aggregate, is orthonormalized using the QR algorithm. The resulting orthonormal basis forms the values of a block column of P^l_{l+1}, while the coefficients representing the old basis with respect to the new basis define B^{l+1}, cf. [23, 20].

NOTE 2.2. In this way, with B^1, A_1, and $\mathbf{b}_1$ given, the entire multigrid setup can be performed. This construction of the algebraic multigrid hierarchy, using (2.6), (2.3), and (2.2), and relying on a *given* fine-level near-kernel representation, B^1, is called the *standard smoothed aggregation* setup in this paper. For later reference, we outline the setup in Algorithm 2 below. For details, see [20]. □

ALGORITHM 2 (Standard smoothed aggregation setup).
Given A_1, B^1, L, do the following for $l = 1, \ldots, L-1$:

(a) *Construct $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$ based on A_l.*
(b) *Construct B^{l+1} and P_{l+1}^l using (2.6) based on $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$.*
(c) *Construct the smoothed prolongator: $I_{l+1}^l = S_l P_{l+1}^l$.*
(d) *Construct the coarse matrix: $A_{l+1} = (I_{l+1}^l)^T A_l I_{l+1}^l$.*

With our choice of smoothing components and a coarsening procedure utilizing (2.6), the standard smoothed aggregation scheme can be proven to converge under certain assumptions on the near-kernel components alone. The following such result motivates the need for standard smoothed aggregation to have access to the near-kernel components and serves to motivate and guide our development of αSA.

Let $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{A}}$ denote the Euclidean inner product over the degrees of freedom corresponding to an agglomerate $\mathcal{A}$ and denote the A_1-norm by $|||\mathbf{u}||| = \langle A_1\mathbf{u}, \mathbf{u}\rangle^{1/2}$. Let B^1 denote an $n_1 \times r$ matrix whose columns are thought to form a basis for the near-kernel components corresponding to A_1.

THEOREM 2.3 (Theorem 4.2 of [20]). *With $\tilde{\mathcal{A}}_i^l$ denoting the set of fine-level degrees of freedom corresponding to aggregate $\mathcal{A}_i^l$ on level l, assume that there exists constant $C_a > 0$ such that, for every $\mathbf{u} \in \mathbb{R}^{n_1}$ and every $l = 1, \ldots, L-1$, the following approximation property holds:*

$$\sum_i \min_{\mathbf{w} \in \mathbb{R}^r} \|\mathbf{u} - B^1\mathbf{w}\|^2_{\tilde{\mathcal{A}}_i^l} \leq C_a \frac{9^{l-1}}{\rho(A_1)} \langle A_1\mathbf{u}, \mathbf{u}\rangle. \tag{2.7}$$

Then

$$|||\mathbf{x}^* - \mathbf{AMG}(\mathbf{x}, \mathbf{b}_1)||| \leq \left(1 - \frac{1}{c(L)}\right) |||\mathbf{x}^* - \mathbf{x}||| \qquad \forall \mathbf{x} \in \mathbb{R}^{n_1},$$

where $A_1\mathbf{x}^* = \mathbf{b_1}$ and $c(L)$ is a polynomial of degree 3 in L.

Since the use of (2.6) is assumed, condition (2.7) reflects an assumption on all tentative prolongators P_{l+1}^l and can be equivalently restated as

$$\sum_i \min_{\mathbf{w} \in \mathbb{R}^r} \|\mathbf{u} - P_2^1 P_3^2 \ldots P_{l+1}^l B^{l+1}\mathbf{w}\|^2_{\tilde{\mathcal{A}}_i^l} \leq C_a \frac{9^{l-1}}{\rho(A_1)} \langle A_1\mathbf{u}, \mathbf{u}\rangle, \tag{2.8}$$

for every $\mathbf{u} \in \mathbb{R}^{n_1}$ and every $l = 1, \ldots, L-1$. Thus, in the context of smoothed aggregation, condition (2.7) can be viewed as an alternative formulation of the weak approximation property [2]. Note that the required approximation of a fine-level vector is less stringent for coarser levels. Also, convergence is guaranteed even though no regularity assumptions have been made. Although the convergence bound naturally depends on the number of levels, computational experiments suggest that the presence of elliptic regularity for standard test problems yields optimal performance (i.e., convergence with bounds that are independent of the number of levels).

That polynomial $c(L)$ in the convergence estimate has degree 3 is an artifact of the proof technique used in [20], where no explicit assumptions are made on the smoothness of the coarse-level bases; instead, only the smoothness guaranteed by application of the simple prolongation smoother, S_l, is considered.

Notice that this convergence result hinges on the selection of B^1. In particular, B^1, and the coarse operators, B^{l+1} and P^l_{l+1}, $1 \le l \le L-1$, that it induces, must guarantee that the left side of (2.8) is small for any vector $\mathbf{u}$ for which $\langle A_1\mathbf{u}, \mathbf{u}\rangle$ is small. Since the standard SA method requires that matrix B^1 be given as input, with the columns of B^1 representing a basis of (a superset of) the near-kernel of A_1, then the construction of P^l_{l+1} in (2.6) guarantees that all coarse-level representations, B^l, form a basis for (a superset of) the near-kernel of A_l, $l > 1$. The purpose of this paper is to enrich a given, incomplete (possibly even empty) set of near-kernel components with approximations computed at runtime in such a way that good convergence can be recovered. The adaptive method can then be viewed as an iterative attempt to satisfy (2.8) heuristically (see Note 3.2 below). Our B^1 is computed only approximately, which means that the coarse-level B^l obtained by (2.6) alone may not be the optimal representation of the near-kernel. To remedy this, we carry out the setup computation also on the coarse levels to improve on the initial guess for the coarse-level candidates given by (2.6).

The following section describes our basic approach to achieving this objective.

3. Self-Correcting Adaptive Setup.

In this section, we describe a prototype of the adaptive method. To maintain generality and simplicity, the discussion is intentionally vague about the various processes involved in the algorithm. Details are provided in Section 4. Suffice it to say here that the coarsening processes are closely related to those used in standard SA.

Before describing the algorithm, we introduce the following notational conventions. The transfer operators and coarse-level problems, as well as other components of our multigrid scheme, change as our method evolves. Whenever possible, we use the same symbols for the updated components. Thus, symbol B^l may denote a single vector in one cycle of the setup procedure, or perhaps a two-column matrix in the next step of the setup. The intended meaning should be clear from context.

3.1. Initialization setup stage.

The adaptive multigrid setup procedure considered in this paper can be split into two stages. If no knowledge of the near-kernel components of A_1 is available, then we start with the first stage to determine an approximation to one such component. This stage also determines the number of levels, L, to be used in the coarsening process. (Changing L in the next stage based on observed performance is certainly possible, but it is convenient to fix L–and other constructs–early in the setup phase.)

Let $\varepsilon > 0$ be a given convergence tolerance.

Algorithm 3 (Initialization stage).

1. *Set $l = 1$, select a random vector, $\mathbf{x}_1 \in \mathbb{R}^{n_1}$, and create copy, $\hat{\mathbf{x}}_1 \leftarrow \mathbf{x}_1$.*

2. *With initial approximation $\mathbf{x}_1$, relax μ times on $A_1\mathbf{x} = 0$:*

$$\mathbf{x}_1 \leftarrow (I - R_1A_1)^\mu \mathbf{x}_1.$$

3. *If $\left(\frac{\langle A_1\mathbf{x}_1, \mathbf{x}_1\rangle}{\langle A_1\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_1\rangle}\right)^{1/\mu} \le \varepsilon$, then set $L = 1$ and* **stop** *(problem $A_1\mathbf{x} = \mathbf{b}_1$ can be solved fast enough by relaxation alone, so only one level is needed).*

4. *Otherwise, do the following:*

(a) *Set* $B^l \leftarrow \mathbf{x}_l$.

(b) *Create a set,* $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$, *of nodal aggregates based on matrix* A_l.

(c) *Define tentative prolongator* P_{l+1}^l *and candidate matrix* B^{l+1} *using the candidate matrix* B^l *and relations (2.6) with structure based on* $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$.

(d) *Define the prolongator:* $I_{l+1}^l = S_l P_{l+1}^l$.

(e) *Define the coarse matrix:* $A_{l+1} = (I_{l+1}^l)^T A_l I_{l+1}^l$. *If level* $l+1$ *is coarse enough that a direct solver can be used there, skip to Step 5; otherwise, continue.*

(f) *Set the next-level approximation vector:* $\mathbf{x}_{l+1} \leftarrow B^{l+1}$.

(g) *Make a copy of the current approximation:* $\hat{\mathbf{x}}_{l+1} \leftarrow \mathbf{x}_{l+1}$.

(h) *With initial approximation* $\mathbf{x}_{l+1}$, *relax* μ *times on* $A_{l+1}\mathbf{x} = 0$*:*

$$\mathbf{x}_{l+1} \leftarrow (I - R_{l+1}A_{l+1})^{\mu}\mathbf{x}_{l+1}.$$

(i) *If* $\left(\frac{\langle A_{l+1}\mathbf{x}_{l+1}, \mathbf{x}_{l+1}\rangle}{\langle A_{l+1}\hat{\mathbf{x}}_{l+1}, \hat{\mathbf{x}}_{l+1}\rangle}\right)^{1/\mu} \leq \varepsilon$, *skip Steps (f-i) in further passes through Step 4.*

(j) *Increment* $l \leftarrow l+1$ *and return to Step 4(a).*

5. *Set* $L \leftarrow l+1$ *and update the finest-level candidate matrix:*

$$B^1 \leftarrow I_2^1 I_3^2 \dots I_{L-1}^{L-2}\mathbf{x}_{L-1}.$$

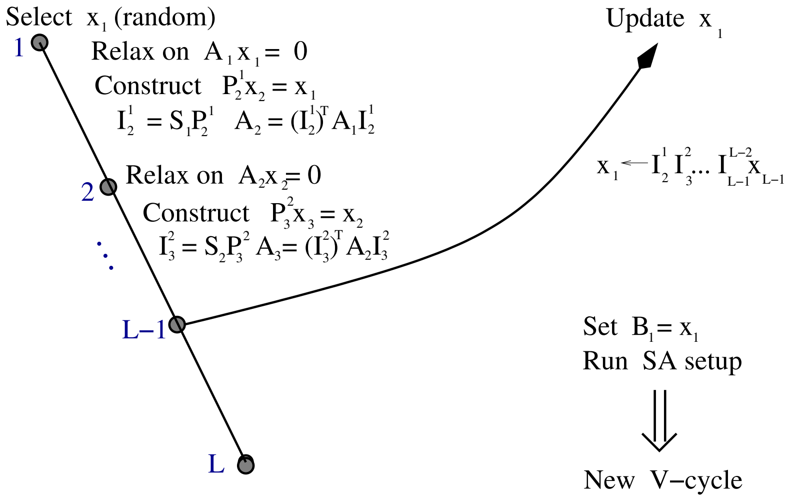
6. *Create the* V*-cycle based on* B^1 *using the standard smoothed aggregation setup of Algorithm 2, with the exception that the aggregates are predetermined in Step 4.*

This initialization stage terminates whenever a level is reached in the coarsening process where a direct solver is appropriate. It does not involve level L processing because it is assumed that the coarsest level is handled by a direct solver, making the stopping criterion in Step 4(i) automatically true. Notice that the candidate matrix is actually a vector in this initial stage because we are computing only one candidate. Note that this stage provides all of the components needed to construct our initial solver, $\mathbf{AMG}_1$.

If the criterion tested in Step 4(i) is satisfied, we are assured that the current coarse level, $l+1$ can be easily solved by relaxation alone. At that point, we could choose not to coarsen further and use relaxation as a coarsest-level solver. However, it is possible that the general stage of the algorithm described below adds more candidates. In case a new candidate approximates the low-energy modes of the problem better than the candidate obtained in the initial step, the coarse-level matrix may no longer be easily solved by relaxation alone. Thus, we choose to coarsen further, until we are sure that the coarsest problem can be handled well. This offers an added benefit of producing, at the end of the initial stage, a complete aggregation that can be reused in the general stage. Note that if 4(i) is satisfied, then the approximate solution of the homogenous problem may be zero. In such a case, we restore the saved original vector $\hat{\mathbf{x}}_{l+1}$. We choose to skip the Steps 4(f-i) in further coarsening once 4(i) is satisfied. This amounts to using standard SA coarsening from level $l+1$ down, which guarantees that the candidate computed on level l is exactly represented all the way to the coarsest level. Figure 3.1 illustrates Algorithm 3.

Note 3.1. The initialization stage described in Algorithm 3 is used only if no knowledge of the near-kernel components is provided. In many situations, however, some knowledge may be available and should be used. In such cases, the initialization stage can be omitted and the initial B^1 can be assumed to consist of the given set

FIG. 3.1. *Initialization stage, Algorithm 3*



of vectors. The initial V-cycle would then be constructed exactly as in Algorithm 2 prior to running the main adaptive setup.

As an example, consider a problem of 3D linear elasticity discretized by a standard linear first-order finite element method over an unstructured grid. In this case, if the discretization package either generates the rigid body modes or supplies the nodal geometry to the solver, then the full set of nullspace vectors are presumably available [22] and the adaptive process may be unnecessary. Otherwise, when the full set of rigid body modes is unavailable, it is nevertheless often possible to obtain a subset of the rigid body modes consisting of three independent constant displacements, regardless of the geometry of the grid. Such a subspace should be used whenever possible to create B^1 and to set up a V-cycle exactly as in the standard smoothed aggregation method. The initialization stage would then be omitted. □

Thus, the initialization stage given by Algorithm 3 should be viewed as optional, to be done only if no information can be assumed about the system to be solved. In view of Note 3.1, we can in any case assume that the initial B^1 has at least one column and that a tentative V-cycle is available. This means that we have constructed aggregates $\mathcal{A}_i^l$, transfer operators P_{l+1}^l and I_{l+1}^l, and coarse operators A_{l+1}, $l = 1, \dots, L-1$.

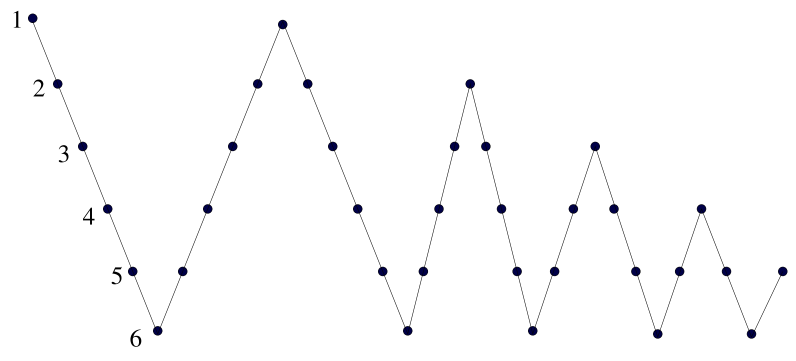
3.2. General setup stage. In each step of the second stage of the adaptive procedure, we apply the current V-cycle to the homogenous problem to uncover error components that are not quickly attenuated. The procedure then updates its own transfer operators to ensure that these components will be eliminated by the improved method, while preserving the previously established approximation properties. Thus, this stage essentially follows the initialization stage with relaxation replaced by the current V-cycle.

One of the subtleties of this approach lies in the method's attempt to update each level of the evolving V-cycle as soon as its ineffectiveness is exposed. Thus, on the finest level in the second stage, the current V-cycle simply plays the role of relaxation: if it is unable to quickly solve the homogeneous problem (i.e., Step 3 fails), then the resulting error becomes a new candidate and new degrees of freedom are generated accordingly on level 2 (i.e., columns are added to B^1). The level 2-to-L part of the old V-cycle (i.e., the part without the finest level) then plays the role of the level 2 relaxation in the initial setup phase and is thus applied to the homogeneous problem to assess the need to improve its coarser-level interpolation operators. The same is done on each coarser level, l, with the level l-to-L part of the old V-cycle playing the role of the level l relaxation step in the initial setup phase. The process continues until adequate performance is observed or the maximum permitted number of degrees of freedom per node is reached on coarse levels.

We present a general prototype algorithm for the adaptive multigrid setup, assuming that a tentative V-cycle has previously been constructed (cf. Note 3.1). We thus assume that a current hierarchy of nodal aggregates, $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$, and operators $P_{l+1}^l, I_{l+1}^l, A_{l+1}$, are available for all $l = 1, \ldots, L-1$.

Consider a method in which, within each cycle of the adaptive setup, we attempt to update the current V-cycle level by level. One cycle of this adaptive setup traverses from the finest to the coarsest level; on each level l along the way, it updates B^l based on computing a new candidate from the current multigrid scheme applied to the homogenous problem on level l. Thus, on level l in the setup process, a solver is applied that traverses from that level to level L and back. This gives us the picture of a backward FMG cycle, where the setup traverses from the finest to the coarsest grid and each level along the way is processed by a V-cycle solver (see Figure 3.2). Now, once this new candidate is computed, it is incorporated into the current multigrid scheme and the previously existing V-cycle components are overwritten on level $l+1$, but temporarily retained from that level down. As a result, we redefine level-by-level the V-cycle components. Once the new B^l (and I_{l+1}^l in (2.3)) are constructed all the way to the coarsest level, we can then use them to update the current B^1 and, based on it, construct a new V-cycle on the finest level.

FIG. 3.2. *Self-correcting adaptive cycling scheme given by Algorithm 4, with the solver cycles uncollapsed.*



The general stage is therefore analogous to the initialization stage described in Algorithm 3, with relaxation replaced by the evolving V-cycle. Instead of using simple relaxation on each level as the initialization stage does, the general stage uses the current solver to identify new types of error that the earlier sweeps of the setup cycle may have missed. The initialization stage was designed to capture a type of error that relaxation cannot efficiently eliminate. A prototype of this algebraically smooth error is the candidate that is generated by applying relaxation to the homogeneous problem on each level. Similarly, each cycle of the general stage is designed to capture a type of error that the current V-cycle cannot handle, and this too must be done on each level. It is important to note that we are talking here about error *type*. It is not enough for the coarsening process to eliminate only the particular candidate: typically, a fixed percentage of the spectrum of A_1 is algebraically smooth, so elimination of one candidate at a time would take $O(n_1)$ setup cycles. Thus, to avoid this unacceptably large cost, each setup cycle must determine interpolation operators so that the solver eliminates a relatively large set of errors of each candidate's type. Just as each rigid body mode is used locally in standard SA to treat errors of similar type (constants represent errors that are smooth within variables and rotations represent inter-variable "smoothness"), so too must each candidate be used in αSA. Moreover, a full set of types must be determined if the solver is to attain full efficiency (e.g., for 2D linear elasticity, three rigid body modes are generally needed). We thus think of each candidate as a sort of *straw man* that represents a class of smooth components. Efficient computation of a full set of straw men is the responsibility of the adaptive process. However, proper treatment of each straw man is the task of the basic solver, which is SA in this case.

As we apply our current method to the homogenous problem, the resulting candidate, $\mathbf{x}_l$, becomes rich in the components of the error that are slow to converge in the current method. Our goal in designing the adaptive algorithm is to ensure that $\mathbf{x}_l$ is approximated relatively well by the newly constructed transfer operator. That is, we want to control the constant C_a in the inequality

$$\min_{\mathbf{v}\in\mathbb{R}^{n_{l+1}}} \|\mathbf{x}_l - P_{l+1}^l \mathbf{v}\|^2 \le \frac{C_a}{\rho(A_l)} \|\mathbf{x}_l\|_{A_l}^2. \tag{3.1}$$

The transfer operators must therefore be constructed to give accurate approximations to each candidate as it is computed. This can be guaranteed locally by requiring that, over every aggregate $\mathcal{A}$, we have

$$\min_{\mathbf{v}\in\mathbb{R}^{n_{l+1}}} \|\mathbf{x}_l - P_{l+1}^l \mathbf{v}\|_{\mathcal{A}}^2 \le C_a \delta_{\mathcal{A}}(\mathbf{x}_l), \tag{3.2}$$

where $\delta_{\mathcal{A}}$ are chosen so that summing (3.2) over all aggregates leads to (3.1), i.e., so that

$$\sum_{\mathcal{A}} \delta_{\mathcal{A}}(\mathbf{x}) = \frac{\langle A_l \mathbf{x}, \mathbf{x}\rangle}{\rho(A_l)}. \tag{3.3}$$

For now, the only assumption we place on $\delta_{\mathcal{A}}(\mathbf{x})$ is that (3.3) holds. An appropriate choice for the definition of $\delta_{\mathcal{A}}(\mathbf{x})$ is given in Section 4.

NOTE 3.2 (Relationship to theoretical assumptions). To relate condition (3.1) to the theoretical foundation of smoothed aggregation, we make the following observation. If P_{l+1}^l is constructed so that (3.1) is satisfied for the candidate $\mathbf{x}_l$, the construction of our method automatically guarantees that

$$\min_{\mathbf{v}\in\mathbb{R}^{n_{l+1}}} \|\mathbf{x}_1 - P_2^1 P_3^2 \dots P_{l+1}^l \mathbf{v}\|^2 \le \frac{C_a}{\rho(A_l)} \|\hat{\mathbf{x}}_l\|_{A_1}^2, \tag{3.4}$$

where $\mathbf{x}_1 = P_2^1 P_3^2 \dots P_l^{l-1} \mathbf{x}_l$ and $\hat{\mathbf{x}}_1 = I_2^1 I_3^2 \dots I_l^{l-1} \mathbf{x}_l$. Since it is easy to show that $\|\hat{\mathbf{x}}\|_{A_1} \le \|\mathbf{x}\|_{A_1}$, we can thus guarantee that (2.8) holds for the particular fine-level candidate $\mathbf{x}_1$. Inequality (3.1) is easily satisfied for any component $\mathbf{u}$ for which $\|\mathbf{u}\|_{A_1}$ is bounded away from zero. We can thus focus on the troublesome subspace of components with small energy. Our experience with the standard SA method indicates that for the 2nd and 4th order elliptic problems it suffices to ensure that the components corresponding to the nullspace of the weak form of the problem are well approximated by the prolongation (the near-nullspace components are then well appoximated due to the localization and smoothing procedures involved in constucting the smoothed aggregation transfer operators). Further, as the set of candidates constructed during the setup cycle is expected to eventually encompass the entire troublesome subspace, satisfaction of (3.1) for all candidates would imply the satisfaction of (2.8) for any $\mathbf{u} \in \mathbb{R}^{n_1}$. This, in turn, guarantees convergence. □

NOTE 3.3 (Locally small components). Each new candidate is the result of applying the V-cycle based on the current B^1, so it must be approximately A_1-orthogonal to all previously computed candidates. This is, however, only a global property that the evolving candidates tend to exhibit. It may be that a candidate is so small on some aggregate, relative to its energy, that its representation there can be ignored. More precisely, we could encounter situations in which

$$\|\mathbf{x}_l\|_{\mathcal{A}}^2 \leq C_a \delta_{\mathcal{A}}(\mathbf{x}_l) \tag{3.5}$$

for a particular aggregate, $\mathcal{A}$, meaning that (3.2) is automatically satisfied, no matter what choice we make for P_{l+1}^l. We can, therefore, test for this condition for each candidate on every aggregate. When the test is positive, we can simply remove the candidate's segment from consideration in construction of that aggregate's transfer operator. This elimination can help control coarse-level complexity since small candidate segments are prevented from generating additional columns of P_{l+1}^l and I_{l+1}^l. (This test should be used in the initialization as well as the general setup stage. We did not include it there for simplicity and because there is generally no worry about complexity in the initial stage.) □

NOTE 3.4 (Construction of P_{l+1}^l to minimize the number of columns). Although each candidate's segments may be too large to ignore, it may be that a nontrivial linear combination of them is small. Thus, we also need to use (3.5) to identify a minimal local basis for constructing the transfer operators so that the global approximation property is maintained. To this end, let subscript $\mathcal{A}$ denote the restriction of the corresponding vector or matrix to the degrees of freedom in $\mathcal{A}$ and let $r_{\mathcal{A}}$ denote the number of columns of $B_{\mathcal{A}}^l$. One possibility for constructing the updated transfer operator, P_{l+1}^l, aggregate by aggregate, would then proceed as follows:

- Rescale each column, $\mathbf{y}$, of B^l globally: $\mathbf{y} \leftarrow \frac{\mathbf{y}}{\sqrt{\langle A_l \mathbf{y}, \mathbf{y}\rangle}}$.
- Reorder the newly scaled columns of $B_{\mathcal{A}}^l$ so that their Euclidean norms over aggregate $\mathcal{A}$ are nonincreasing: $\|\mathbf{y}_1\|_{\mathcal{A}} \geq \|\mathbf{y}_2\|_{\mathcal{A}} \geq \ldots \geq \|\mathbf{y}_{r_{\mathcal{A}}}\|_{\mathcal{A}}$.
- Set $j = 1$.
- While $j \leq r_{\mathcal{A}}$:
  - Set $\gamma_{\mathcal{A}} = \frac{C_a \delta_{\mathcal{A}}(\mathbf{y}_j)}{\langle A_l \mathbf{y}_j, \mathbf{y}_j\rangle}$.
  - If $\|\mathbf{y}_j\|_{\mathcal{A}}^2 \leq \gamma_{\mathcal{A}}$, then stop. Otherwise, add $\frac{\mathbf{y}_j}{\|\mathbf{y}_j\|_{\mathcal{A}}}$ as a new column of P_{l+1}^l, make all remaining columns in $B_{\mathcal{A}}^l$ orthogonal to $\mathbf{y}_j$, and reorder them so that their Euclidean norms over $\mathcal{A}$ are nonincreasing.
  - $j \leftarrow j + 1$.

A disadvantage of this process is that P_{l+1}^l (hence, also I_{l+1}^l) must, in principle, be constructed from scratch in each cycle of the adaptive setup. We discuss other practical issues associated with this approach in Note 4.2. □

NOTE 3.5 (Reusing previously constructed components). To exploit the work done in the earlier steps of the setup as much as possible, we consider an alternate procedure that reuses parts of P_{l+1}^l that have already been computed. Thus, in each step of the setup, we only consider adding a single new column to P_{l+1}^l. This has the advantages that less work is required and that the storage used to hold the global candidates can be reused as soon as they have been incorporated into P_{l+1}^l.

In this approach, to minimize the complexity of the transfer operators, we seek to ignore locally those components of candidate $\mathbf{x}_l$ that appear to be well approximated by the current transfer operators. This includes the case when $\mathbf{x}_l$ is locally small in the sense of (3.5). To decide whether to ignore $\mathbf{x}_l$ locally in the construction of *new* tentative prolongator P_{l+1}^l, we test how well it is approximated by the *current* tentative prolongator, $\tilde{P}_{l+1}^l$. The following provides a test of how well the range of $\tilde{P}_{l+1}^l$ approximates $\mathbf{x}_l$ over aggregate $\mathcal{A}$:

$$\|\mathbf{x}_l - \tilde{P}_{l+1}^l(\tilde{P}_{l+1}^l)^T\mathbf{x}_l\|_{\mathcal{A}}^2 \le C_a \delta_{\mathcal{A}}(\mathbf{x}_l). \tag{3.6}$$

(Since $(\tilde{P}_{l+1}^l)^T\tilde{P}_{l+1}^l = I$, then $\tilde{P}_{l+1}^l(\tilde{P}_{l+1}^l)^T$ is the L^2 projection onto the range of $\tilde{P}_{l+1}^l$; thus, (3.6) is just approximation property (3.2) using the tentative prolongator in place of the smoothed one.) If (3.6) is satisfied, then $\mathbf{x}_l$ is assumed to be well approximated by the current transfer operator and is simply ignored in the construction of the new transfer operator on aggregate $\mathcal{A}$. (Practical implications of this local elimination from the coarsening process are considered in Section 4.) If the inequality is not satisfied, then we keep the computed vector $\mathbf{y} = \mathbf{x}_l - \tilde{P}_{l+1}^l(\tilde{P}_{l+1}^l)^T\mathbf{x}_l$, which, by construction, is orthogonal to all the vectors already represented in the current $\tilde{P}_{l+1}^l$. We then normalize via $\mathbf{y} \leftarrow \mathbf{y}/\|\mathbf{y}\|_{\mathcal{A}}$ so that the new P_{l+1}^l has orthonormal columns: $(P_{l+1}^l)^T P_{l+1}^l = I$. □

Before we introduce Algorithm 4 below, we stress that the description should be viewed as a general outline of the adaptive multigrid setup. We intentionally ignore several practical issues that must be addressed before this algorithm can be implemented. For instance, we do not include details on how the new B^l and I_{l+1}^l are efficiently constructed in the evolving method. Also, when using a coarse-level V-cycle constructed by previous applications of the setup stage, we must deal with the possibility that the number of vectors approximated on coarse levels in previous cycles is smaller than the number of vectors approximated on the fine levels in the current cycle. These issues are discussed in Section 4, where we take advantage of the smoothed aggregation framework to turn the prototypical Algorithm 4 into a practical implementation.

Assume we are given a bound, $K \in \mathbb{N}$, on the number of degrees of freedom per node on coarse levels, convergence factor tolerance $\varepsilon \in (0,1)$, and aggregate quantities $\delta_{\mathcal{A}}(x)$ such that $\sum_{\mathcal{A}} \delta_{\mathcal{A}}(x) = \frac{\langle A_l \mathbf{x}, \mathbf{x}\rangle}{\rho(A_l)}$.

ALGORITHM 4 (One cycle of the general setup stage).

1. *If the maximum number of degrees of freedom per node on level 2 equals K,* **stop** *(the allowed number of coarse-grid degrees of freedom has been reached).*

2. *Create a copy of the current B^1 for later use: $\hat{B}^1 \leftarrow B^1$.*

3. *Select a random $\mathbf{x}_1 \in \mathbb{R}^{n_1}$, create a copy $\hat{\mathbf{x}}_1 \leftarrow \mathbf{x}_1$, and apply μ iterations of the current finest-level V-cycle:*

$$\mathbf{x}_1 \leftarrow \mathbf{AMG}_1^{\mu}(\mathbf{x}_1, \mathbf{0}).$$

4. *If $\left(\frac{\langle A_1\mathbf{x}_1,\mathbf{x}_1\rangle}{\langle A_1\hat{\mathbf{x}}_1,\hat{\mathbf{x}}_1\rangle}\right)^{1/\mu} \le \varepsilon$, then* **stop** *($A_1\mathbf{x} = \mathbf{b}_1$ can be solved fast enough by the current method).*

5. *Update* B^1 *by incorporating the computed* $\mathbf{x}_1$ *in its range:*

$$B^1 \leftarrow [B^1, \mathbf{x}_1].$$

6. *For* $l = 1, \ldots, L-2$*:*

(a) *Create a copy of the current* B^{l+1} *for later use:* $\hat{B}^{l+1} \leftarrow B^{l+1}$*,*

(b) *Define new coarse-level matrix* B^{l+1} *and transfer operators* P^l_{l+1}, I^l_{l+1} *using (2.6) and (2.3). In creating* P^l_{l+1}*, some local components in* B^l *may be eliminated as suggested in Notes 3.4 and 3.5.*

(c) *Construct coarse operator* $A_{l+1} = (I^l_{l+1})^T A_l I^l_{l+1}$*.*

(d) *Set coarse-level approximation* $\mathbf{x}_{l+1}$ *to be the last column of* B^{l+1}*.*

(e) *Make a copy:* $\hat{\mathbf{x}}_{l+1} \leftarrow \mathbf{x}_{l+1}$*.*

(f) *Apply* μ *iterations of the current-level* $l+1$ V*-cycle:*

$$\mathbf{x}_{l+1} \leftarrow \mathbf{AMG}^{\mu}_{l+1}(\mathbf{x}_{l+1}, \mathbf{0}).$$

(g) *If* $\left(\frac{\langle A_{l+1}\mathbf{x}_{l+1}, \mathbf{x}_{l+1}\rangle}{\langle A_{l+1}\hat{\mathbf{x}}_{l+1}, \hat{\mathbf{x}}_{l+1}\rangle}\right)^{1/\mu} \leq \varepsilon$*, then skip Steps (d) through (h) in further passes through Step 6.*

(h) *Update* B^{l+1} *by ensuring that the newly computed* $\mathbf{x}_{l+1}$ *is in its range:*

$$B^{l+1} \leftarrow [\hat{B}^{l+1}, \mathbf{x}_{l+1}].$$

7. *Update the finest-level candidate:*

$$\mathbf{x}_1 \leftarrow I^1_2 I^2_3 \ldots I^{L-2}_{L-1} \mathbf{x}_{L-1}. \tag{3.7}$$

8. *Update* B^1 *by adding the newly computed* $\mathbf{x}_1$ *to the range of* $\hat{B}^1$*:*

$$B^1 \leftarrow [\hat{B}^1, \mathbf{x}_1].$$

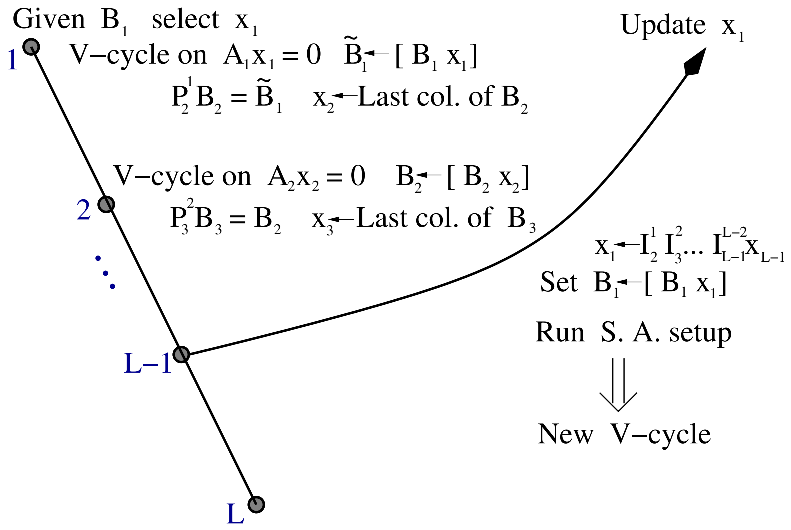
9. *Create a* V*-cycle,* **AMG***, based on* B^1 *using the standard smoothed aggregation setup of Algorithm 2.*

Algorithm 4, which is illustrated in Figure 3.3, starts from a V-cycle on input and produces an improved V-cycle as output. It stops iterating when either the convergence factor for the fine-level iteration in Step 3 is acceptable (as measured in Step 4) or when the maximum number of iterations is reached. Note that, as with the initial stage, this general stage does not involve level L processing because the coarsest level is assumed to be treated by a direct solver. Also as in the initial stage, once a level is reached where the problem can be solved well by the current method, any further coarsening is constructed as in the standard SA.

4. Implementation Issues. Several issues must be addressed to make Algorithm 4 practical. We take advantage of certain features of the smoothed aggregation concept to carry out the method outlined in Algorithm 4, as well as to control the amount of work required to keep the evolving coarse-level hierarchy up to date.

As suggested in Notes 3.4 and 3.5, a candidate may occasionally be eliminated locally over an aggregate. This results in varying numbers of degrees of freedom per node on the coarse levels. (Recall that a coarse-level node is defined as a set of degrees of freedom, each representing the restriction of a single candidate to a fine-level aggregate.) To simplify notation, we assume for the time being that the number of degrees of freedom per node is the same for all nodes on a given level (i.e., no

FIG. 3.3. *One step of general setup stage, Algorithm 4.*



candidates are locally eliminated). It is important, however, to keep in mind that we are interested in the more general case. A generalization to varying numbers of degrees of freedom per node could be obtained easily at the cost of a much more cumbersome notation. We briefly remark on the more general cases in Notes 4.2 and 4.3 below.

NOTE 4.1 (Construction of temporary "bridging" transfer operators). An issue we must consider is the interfacing between the emerging V-cycle on finer levels and the previous V-cycle on coarser levels. Each setup cycle starts by selecting an initial approximation for a new candidate on the finest level (cf. Figure 3.3). This approximation is then improved by applying the error propagation matrix for the previously constructed V-cycle to it. The resulting candidate is used to enrich B^1. This necessitates an update of P^1_2, I^1_2, and A_2 from (2.6) and (2.2), and introduces an additional degree of freedom for the nodes on level 2. Since we now want to run the current solver on level 2 to obtain an improved candidate on that level, we need to temporarily modify P^2_3 and I^2_3 because these transfer operators have not yet been updated to reflect the added degrees of freedom on level 2. Once this modification has been made, a V-cycle on level 2 can be run to compute the new candidate there. This candidate is then incorporated into B^2 and new P^2_3 and I^2_3 are constructed, overwriting the temporary versions, and the new A_3 can be computed using (2.2). To perform the V-cycle on level 3, we then must temporarily modify operators P^3_4 and I^3_4 for the same reason we had to update P^2_3 and I^2_3 above. Analogous temporary modifications to the transfer operators are necessary on all coarser levels, as the setup cycle traverses sequentially through them.

Thus, on stage l of a single cycle of the setup process, all transfer operators defining the V-cycle can be used without change, except for P^l_{l+1} and, consequently, I^l_{l+1} defined through (2.3). We can construct the temporary operator, P^l_{l+1}, by modifying (2.6) as

$$P^l_{l+1} B^{l+1} = \hat{B}^l,$$

where $\hat{B}^l$ is formed by removing the last column from B^l, which consists of the

$k+1$ fine-level candidate vectors, including the newly added one (so that the first k candidates are the same as in the previous cycle). Since tentative prolongator P_{l+1}^l produced in this way is based only on fitting the first k vectors in B^l, the coarse-level matrix A_{l+1} resulting from the previous cycle of the αSA setup (described below) can be used on the next level. Thus, all the coarse operators for levels coarser than l can be used without change. This has the advantage of reducing the amount of work to keep the V-cycle up to date on coarser, yet-to-be-traversed levels. □

So far, we have only considered the case where all candidates are used locally. In the interest of keeping only the candidates that are essential to achieving good convergence properties, we now consider locally eliminating the candidates where appropriate.

NOTE 4.2 (Eliminating candidates locally as suggested in Note 3.5). When we eliminate a candidate locally over an aggregate as suggested in Note 3.5, the construction of the bridging operator above can be easily modified so that the multigrid hierarchy constructed in the previous setup cycle can be used to apply a level l V-cycle in the current one. Since the procedure guarantees that the previously selected candidates are retained and only the newly computed candidate may be locally eliminated, the V-cycle constructed in the previous setup cycle remains valid on coarser grids as in the case of Note 4.1. The only difference now is that aggregates may have a variable number of associated candidates, and the construction of the temporary transfer operator P_{l+1}^l described in Note 4.1 must account for this when removing the column of B^l to construct $\hat{B}^l$. □

NOTE 4.3 (Eliminating candidates locally as suggested in Note 3.4). The situation is slightly more complicated when the procedure described in Note 3.4 is used to eliminate candidates locally over an aggregate. First, even if none of the old candidates are eliminated, the use of the procedure of Note 3.4 may result in a permutation of the candidates over an aggregate, hence a permutation of the coarse degrees of freedom corresponding to the associated node. To match the fine-level V-cycle with the existing coarser levels, an appropriate permutation of the coarse degrees of freedom must then be done when performing the intergrid transfer in the application of the resulting V-cycle.

However, if some of the previously selected candidates are eliminated in favor of the new candidate in the construction of the updated P_{l+1}^l, the coarse V-cycle should no longer be used without change. In such cases, we would have to generate all the coarse levels below level l before running the level $l+1$ V-cycle. This results in a significantly increased cost of the setup phase. □

NOTE 4.4 (Selection of the local quantities $\delta_{\mathcal{A}}(\mathbf{x})$). Our algorithm relies on local aggregate quantities $\delta_{\mathcal{A}}(\mathbf{x})$ to decide whether to eliminate candidate $\mathbf{x}$ in aggregate $\mathcal{A}$, and to guarantee that the computed candidates satisfy the global approximation property (3.1). This leads us to the choice

$$\delta_{\mathcal{A}}(\mathbf{x}) = \left(\frac{\mathrm{card}(\mathcal{A})}{N_l}\right)\frac{\langle A_l\mathbf{x},\mathbf{x}\rangle}{\rho(A_l)}, \tag{4.1}$$

where $\mathrm{card}(\mathcal{A})$ denotes the number of nodes in aggregate $\mathcal{A}$ on level l, and N_l is the total number of nodes on that level. Note that $\sum_{\mathcal{A}}\delta_{\mathcal{A}}(\mathbf{x}) = \frac{\langle A_l\mathbf{x},\mathbf{x}\rangle}{\rho(A_l)}$ for any $\mathbf{x}$, so this can be used in local estimates (3.2) to guarantee (3.1). □

Suppose that we are given a bound, $K > 0$, on the maximum allowed number of degrees of freedom per node on the coarse levels, and a tolerance, $\varepsilon \in (0,1)$, on the target convergence factor. Then one adaptive setup cycle is defined as follows.

Algorithm 5 (One cycle of αSA).

1. *If the maximum number of degrees of freedom per node on level 2 equals K,* **stop** *(the allowed number of coarse grid degrees of freedom has been reached).*

2. *Create a copy of the current B^1 for later use: $\hat{B}^1 \leftarrow B^1$.*

3. *Select a random $\mathbf{x}_1 \in \mathbb{R}^{n_1}$, create a copy $\hat{\mathbf{x}}_1 \leftarrow \mathbf{x}_1$, and apply μ iterations of the current V-cycle:*

$$\mathbf{x}_1 \leftarrow \mathbf{AMG}_1^{\mu}(\mathbf{x}_1, \mathbf{0}).$$

4. *If $\left(\frac{\langle A_1\mathbf{x}_1, \mathbf{x}_1\rangle}{\langle A_1\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_1\rangle}\right)^{1/\mu} \le \varepsilon$, then* **stop** *($A_1\mathbf{x} = \mathbf{b}_1$ can be solved fast enough by the current method).*

5. *Update B^1 by extending its range with the new column $\{\mathbf{x}_1\}$:*

$$B^1 \leftarrow [B^1, \mathbf{x}_1].$$

6. *For $l = 1, \ldots, L-2$:*

- **(a)** *Define a new coarse-level matrix B^{l+1} and transfer operator P^l_{l+1} based on (2.6), using B^l and decomposition $\{\mathcal{A}^l_i\}_{i=1}^{N_l}$. In creating P^l_{l+1}, some local components in B^l may be locally eliminated as suggested in Note 3.5.*
- **(b)** *Construct the prolongator: $I^l_{l+1} = S_l P^l_{l+1}$.*
- **(c)** *Construct the coarse operator: $A_{l+1} = (I^l_{l+1})^T A_l I^l_{l+1}$.*
- **(d)** *Reorder the columns of B^{l+1} so that its last is $\mathbf{x}_{l+1}$ and let $\hat{B}^{l+1}$ consist of all other columns of B^{l+1}.*
- **(e)** *Create a "bridge" transfer operator P^{l+1}_{l+2} to the coarser level with the old B^{l+1} by fitting all the vectors in B^{l+1} except the last one; see Note 4.1.*
- **(f)** *Set the new "bridging" prolongator: $I^{l+1}_{l+2} = S_{l+1} P^{l+1}_{l+2}$.*
- **(g)** *Make a copy: $\hat{\mathbf{x}}_{l+1} \leftarrow \mathbf{x}_{l+1}$.*
- **(h)** *Apply μ iterations: $\mathbf{x}_{l+1} \leftarrow \mathbf{AMG}^{\mu}_{l+1}(\mathbf{x}_{l+1}, \mathbf{0})$.*
- **(i)** *If $\left(\frac{\langle A_{l+1}\mathbf{x}_{l+1}, \mathbf{x}_{l+1}\rangle}{\langle A_{l+1}\hat{\mathbf{x}}_{l+1}, \hat{\mathbf{x}}_{l+1}\rangle}\right)^{1/\mu} \le \varepsilon$, then skip (d) through (j) in further passes through Step 6.*
- **(j)** *Update the coarse representation of candidate B^{l+1}:*

$$B^{l+1} \leftarrow [\hat{B}^{l+1}, \mathbf{x}_{l+1}].$$

7. *Update the latest fine-level candidate:*

$$\mathbf{x}_1 \leftarrow I^1_2 I^2_3 \ldots I^{L-2}_{L-1} \mathbf{x}_{L-1}. \tag{4.2}$$

8. *Update B^1 by extending the old copy with the newly computed $\mathbf{x}_1$:*

$$B^1 \leftarrow [\hat{B}^1, \mathbf{x}_1].$$

9. *Create the V-cycle based on the current B^1 using the standard smoothed aggregation setup described by Algorithm 2.*

Note that if we used the candidate elimination scheme of Note 3.4 in 6(a), we should modify the algorithm to construct a completely new multigrid hierarchy on levels $l+1$ through L before applying the level $l+1$ V-cycle in Step 6(h) (cf. Note 4.2).

Before presenting computational results, we consider several possible improvements intended to reduce the necessary number of cycles of the setup and the amount of work required to carry each cycle.

NOTE 4.5 (Improving the quality of existing candidates). Many practical situations, including fourth-order equations and systems of fluid and solid mechanics, require a set of multiple candidates to achieve optimal convergence. In the interest of keeping operator complexity as small as possible, it is imperative that the number of candidates used to produce the final method be controlled. Therefore, ways of improving the quality of each candidate are of interest, to curb the demand for the growth in their number.

When the current V-cycle hierarchy is based on approximating at least two candidates (in other words, the coarse problems feature at least two degrees of freedom per node), this can be easily accomplished as follows.

Assume that the currently available candidate vectors are $\mathbf{x}_1, \ldots, \mathbf{x}_k$. Consider one such candidate, say, $\mathbf{x_j}$, that we want to improve. We want to run a modified but current V-cycle on the homogenous problem, $A_1\mathbf{x} = \mathbf{0}$, using $\mathbf{x_j}$ as the initial guess. The modification consists of disabling, in the coarse-grid correction process, the columns of the prolongator corresponding to the given candidate. That is, instead of $\mathbf{x}_l \leftarrow \mathbf{x}_l + I_{l+1}^l \mathbf{x}_{l+1}$ in step 2(c) of Algorithm 1, we use

$$\mathbf{x}_l \leftarrow \mathbf{x}_l + I_{l+1}^l \hat{\mathbf{x}}_{l+1},$$

where $\hat{\mathbf{x}}_{l+1}$ is obtained from $\mathbf{x}_{l+1}$ by setting to zero every entry corresponding to fine-level candidate $\mathbf{x_j}$. Thus, the columns of I_{l+1}^l corresponding to $\mathbf{x_j}$ are not used in coarse-grid correction.

In this way, we come up with an improved candidate vector without restarting the entire setup iteration from scratch and without adding a new candidate. Since we focus on one component at a time and keep all other components intact, this modified V-cycle is expected to converge rapidly. □

NOTE 4.6 (Saving work). The reuse of current coarse-level components described in Note 4.1 reduces the amount of work required to keep the V-cycle up to date. Additional work can be saved by performing the decomposition of nodes into disjoint aggregates only during the setup of the initial V-cycle and then reusing this decomposition in later cycles. Yet further savings are possible in coarsening, assuming the candidates are allowed to be locally eliminated according to Note 3.5. For instance, we can exploit the second-level matrix structure:

$$A_2 = \begin{bmatrix} \tilde{A}_2 & X \\ Y & Z \end{bmatrix},$$

where $\tilde{A}_2$ is the second-level matrix from the previous cycle. Thus, A_2 need not be recomputed and can be obtained by a rank-one update of each block entry in $\tilde{A}_2$. In a similar fashion, the new operators P_{l+1}^l, B^{l+1} do not have to be recomputed in each new setup cycle by the local QR decomposition noted in Section 2. Instead, it is possible to update each nodal entry in $\tilde{P}_{l+1}^l, \hat{B}^{l+1}$ by a rank-one update on all coarse levels, where $\tilde{P}_{l+1}^l, \hat{B}^{l+1}$ are the operators created by the previous setup cycle. □

5. Numerical Experiments. To demonstrate the effectiveness of the proposed adaptive setup process, we present results obtained by applying the method to several model problems. In these tests, the solver was stopped when the relative residual reached the value $\varepsilon = 10^{-12}$ (unless otherwise specified). The value $C_a = 10^{-3}$ was used for test (3.6) and the relaxation scheme for the multigrid solver was symmetric Gauss-Seidel. While a Krylov subspace process is used often in practice, we present these results for a basic multigrid V-cycle with no acceleration scheme for clarity, unless explicitly specified otherwise.

All the experiments have been run on a notebook computer with a 1.6 GHz mobile Pentium 4 processor and 512 MB of RAM. For each experiment, we report the following. The column denoted by "Iter" contains the number of iterations required to reduce the residual by the prescribed factor. The "Factor" column reports convergence factor measured as the geometric average of the residual reduction in the last 10 iterations. In the "CPU" column, we report the total CPU times in seconds required to complete both the setup and iteration phases of the solver. In the column "RelCPU" we report the relative times to solution, with one unit defined as the time required to solve the problem given the correct near-nullspace components. In the "OpComp" column, we report the operator complexity associated with the V-cycle for every run (we define operator complexity in the usual sense [19], as the ratio of the number of entries stored in all problem matrices on all levels divided by the number of entries stored in the finest-level matrix.) The "Candidates" column indicates the number of kernel vectors computed in the setup iteration (a value of "provided" means that complete kernel information was supplied to the solver, assuming standard discretization and ignoring scaling). Parameter $\mu_{\max}$ denotes the maximal number of tentative V-cycles allowed in computing each candidate.

In all the cases considered, the problem was modified either by scaling or by rotating each nodal entry in the system by a random angle (as described below). These modifications pose serious difficulties for classical algebraic iterative solvers that are not aware of such modifications, as we assume here.

For comparison, we also include the results for the unmodified problem, with a supplied set of kernel components. Not surprisingly, the standard algorithm (without benefit of the adaptive process) performs poorly for the modified system when the details of this modification are kept from the solver, as we assume here.

We start by considering a diagonally scaled problem:

$$A \leftarrow D^{-1/2} A D^{-1/2},$$

where the original A is the matrix obtained by standard Q1 finite element discretization of the 3D Poisson operator on a cube and D is a diagonal matrix with entries 10^{β}, where $\beta \in [-\sigma, +\sigma]$ is chosen randomly. Table 5.1 shows the results for different values of parameter σ and different levels of refinement. Using the supplied kernel yields good convergence factors for the unmodified problem, but the performance is poor and deteriorates with increased problem size when used with $\sigma \neq 0$. In contrast, the adaptive process, starting from a random approximation, recovers the convergence properties associated with the standard Poisson problem ($\sigma = 0$), even for the scaled case, with convergence that appears independent of the problem size.

The second problem comes from a diagonally scaled matrix arising in 2D elasticity. Diagonal entries of D are again defined as 10^{β}, with $\beta \in [-\sigma, +\sigma]$ chosen randomly. The original matrix is the discrete operator for the plane-strain elasticity formulation over a square domain using bilinear finite elements on a uniform grid, with a Poisson ratio of $\nu = 0.3$ and Dirichlet boundary conditions specified only along the "West" side of the domain. The results in Table 5.2 follow a pattern similar to those for the Poisson problem. Note, however, that more than the usual three candidate vectors are now needed to achieve convergence properties similar to those of the unscaled problem when a correct set of three rigid-body modes is provided by the user. For the scaled problem, however, supplying the rigid-body modes computed based on the problem geometry leads, as expected, to dismal performance of the standard solver.

σ	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
Poisson problem with 68,921 degrees of freedom							
0	provided	N/A	9	0.100	3.65	1.00	1.038
0	1	5	9	0.100	4.09	1.12	1.038
6	provided	N/A	150	0.871	43.76	11.99	1.038
6	1	5	10	0.126	4.27	1.17	1.038
Poisson problem with 1,030,301 degrees of freedom							
0	provided	N/A	9	0.093	58.43	1.00	1.039
0	1	5	9	0.099	80.05	1.37	1.039
6	provided	N/A	690	0.970	3,252.80	55.67	1.039
6	1	5	9	0.096	88.23	1.51	1.039

TABLE 5.1

Misscaled 3D Poisson problems, 68,921 and 1,030,301 degrees of freedom; using $\varepsilon = 10^{-8}$.

The third set of experiments is based again on the 2D elasticity problem, but now each nodal block is rotated by a random angle $\beta \in [0, \pi]$:

$$A \leftarrow Q^T AQ,$$

where Q is a nodal block-diagonal matrix consisting of rotations with random angles. The results in Table 5.3 show that αSA can recover good convergence factors for both the unmodified and the modified systems. Without the adaptive procedure, our basic algebraic solver could not solve the modified matrix problem in a reasonable amount of time.

The fourth example demonstrates performance of the method when a higher number of candidates is required. We consider a 3D elasticity problem with local rotations. This is done to maintain locally orthogonal coordinates, but is otherwise a random rotation of the three degrees of freedom at each node. The model problem we start from is linearized elasticity discretized using trilinear finite elements over a uniform grid. Dirichlet boundary conditions are specified on the "West" face of the cube and the Poisson ratio is set to $\nu = 0.3$. The results in Table 5.4 show that, even for the modified system, the adaptive method can again recover good convergence factors. Furthermore, our current method mimics the convergence of the smoothed aggregation for the unmodified problem with the supplied set of rigid-body modes. In this set of experiments, we can get close to the ideal iteration counts using just 6 candidates. We see that using one extra candidate can improve convergence properties and in some cases actually lower the overall cost of the total time to solution. This is done at the price of a small increase in operator complexity. For problems with multiple right-hand sides, the more expensive setup would be performed only once, and using the extra candidate may then be preferred.

The final example demonstrates performance of the adaptive method for a three-dimensional elasticity problem featuring discontinuities in the Young modulus. Here we consider a 3D elasticity problem in which the Young modulus is allowed to vary randomly between the elements. We consider two cases: a case of coefficients varying randomly with uniform distribution in the interval $(1, 10^{\sigma})$, and the case where the distribution is exponential, i.e., the Young modulus is computed as $10^{(\sigma r)}$, where r is generated randomly with uniform distribution in (0, 1). Keeping with the usual practice of employing Krylov method acceleration for problems with coefficient discontinuities, in this experiment we use our adaptive method as a preconditioner in

σ	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
2D elasticity problem, $80,400$ degrees of freedom							
0	3 provided	N/A	17	0.21	9.16	1.00	1.27
0	3	6	23	0.37	21.16	2.31	1.27
0	3	15	18	0.23	26.65	2.91	1.27
6	3 provided	N/A	299	0.92	133.55	14.58	1.27
6	3	6	25	0.38	22.26	2.43	1.27
6	3	15	18	0.25	27.30	2.98	1.27
2D elasticity problem, $181,202$ degrees of freedom							
0	3 provided	N/A	23	0.35	22.85	1.00	1.28
0	3	15	267	0.937	272.14	11.91	1.27
0	4	15	26	0.422	75.18	3.29	1.50
0	4	20	26	0.439	86.60	3.79	1.50
0	5	15	20	0.314	88.20	3.86	1.78
6	3 provided	N/A	$5,000^{*}$	0.996	4,559.95	199.56	1.28
6	4	15	23	0.367	74.95	3.28	1.50
6	4	20	19	0.302	76.78	3.36	1.50
6	5	10	14	0.173	69.46	3.04	1.78

TABLE 5.2

Scaled 2D elasticity problems with $80,400$ *and* $181,202$ *degrees of freedom. Iteration counts marked with an asterisk indicate that residual reduction by* 10^{12} *was not achieved before the maximum number of iterations was reached.*

the conjugate gradient method. The iteration was stopped once the initial residual was reduced by 10^8. Table 5.5 compares the results obtained by using our adaptive scheme, started from random initial guess, to the results obtained when the method based on a priori knowledge of the rigid body modes is employed as a preconditioner. The table indicates that using the adaptive procedure, without a priori knowledge of the probem geometry, we can about recover the rates of the method based on the knowledge of the rigid-body modes.

The topic of problems with discontinuous coefficients and the appropriate modificiations to the basic smoothed aggregation method will be studied in a separate paper.

NOTE 5.1. The operator complexities in all of the test problems remain below 2. Moreover, for the larger spatial dimension of 3D, these complexities improve somewhat, due largely to the increased speed of aggregation coarsening. It is also worth mentioning that the increasing size of the coarse matrix block entries due to the increasing number of candidates does not significantly impact the time needed to perform one iteration of the solver, apparently due to the more efficient memory access afforded by blocking. □

6. Conclusions. We consider a new multilevel method to tackle problems which, to date, have been very difficult to handle by algebraic multigrid methods. At the expense of a somewhat more costly setup stage and more intricate implementation, we design a method that has, thus far, proved successful at solving such problems. We observe that the convergence properties of the method seem very insensitive to modifications of the algebraic system by scaling or nodal rotation. Moreover, the solver is flexible and can benefit from extra information supplied about the problem. If such information is lacking or incorrect, then αSA can act as a full black-box

Rotated	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
2D elasticity problem with 80,400 degrees of freedom							
NO	3 provided	N/A	17	0.21	9.16 s	1.00	1.27
NO	3	15	18	0.23	26.66	2.91	1.27
YES	3 provided	N/A	1,329	0.99	587.80	64.17	1.27
YES	3	15	19	0.27	27.8464	3.04	1.27
2D elasticity problem with 181,202 degrees of freedom							
NO	3 provided	N/A	23	0.35	22.85 s	1.00	1.28
NO	3	15	18	0.23	66.49	2.91	1.28
YES	3 provided	N/A	$5,000^*$	0.999	3,968.36	173.67	1.28
YES	3	20	135	0.885	170.23	7.45	1.28
YES	4	15	27	0.488	77.46	3.39	1.50
YES	4	20	21	0.395	79.29	3.47	1.50
YES	5	6	18	0.34	60.78	2.66	1.78
YES	5	10	15	0.233	72.66	3.18	1.78

Table 5.3

Rotated 2D elasticity problems with 80,400 *and* 181,202 *degrees of freedom. Iteration counts marked with an asterisk indicate that residual reduction by* 10^{12} *was not achieved before the limit on the number of iterations was reached.*

Rotated	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
3D elasticity problem with 114,444 degrees of freedom							
NO	6 provided	N/A	16	0.20	29.97	1.00	1.159
NO	6	15	20	0.27	189.11	6.31	1.159
NO	7	15	17	0.21	215.78	7.20	1.217
YES	6 provided	N/A	587	0.97	913.49	30.48	1.159
YES	6	15	16	0.22	184.32	6.15	1.159
YES	7	10	15	0.20	171.73	5.73	1.217
YES	7	15	15	0.20	210.99	7.04	1.217
3D elasticity problem with 201,720 degrees of freedom							
NO	6 provided	N/A	16	0.20	50.33	1.00	1.153
NO	6	15	21	0.31	319.60	6.35	1.153
NO	7	10	17	0.216	297.95	5.92	1.209
NO	7	15	17	0.209	363.38	7.22	1.209
YES	6 provided	N/A	739	0.97	1,924.62	38.24	1.153
YES	6	15	16	0.23	308.02	6.12	1.153
YES	7	10	15	0.20	301.98	6.00	1.209
YES	7	15	14	0.16	357.85	7.11	1.209

Table 5.4

Rotated 3D elasticity problems with 114,444 *and* 201,720 *degrees of freedom*

solver. Despite the growing number of degrees of freedom per coarse-level node as the method evolves, the overall cost of one step of the final iteration grows only modestly because of better utilization of cache memory due to dense matrix operations on the nodal blocks. Operator complexity remains at reasonable levels and actually seems to improve with increasing spatial dimension.

The construction of the tentative prolongator in the setup phase involves restric-

σ	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
Elasticity problem with uniformly distributed coeficient jumps							
2	6 provided	N/A	8	0.0734	24.22	1.00	1.15
2	6	10	13	0.2209	219.34	9.05	1.15
2	7	10	11	0.1866	266.29	10.99	1.21
3	6 provided	N/A	8	0.0765	24.23	1.00	1.15
3	6	10	13	0.2154	218.57	9.02	1.15
3	7	10	11	0.1861	265.70	10.96	1.21
4	6 provided	N/A	8	0.0768	24.56	1.00	1.15
4	6	10	12	0.2081	219.26	8.93	1.15
4	7	10	11	0.1648	264.38	10.76	1.21
Elasticity problem with exponentially distributed coeficient jumps							
2	6 provided	N/A	9	0.1146	25.99	1.00	1.15
2	6	10	16	0.3048	225.52	8.68	1.15
2	7	10	12	0.2141	267.72	10.30	1.21
3	6 provided	N/A	14	0.2466	35.68	1.00	1.15
3	6	10	22	0.4179	237.56	6.76	1.15
3	7	10	16	0.3095	275.62	7.84	1.21
4	6 provided	N/A	20	0.3948	49.99	1.00	1.15
4	6	10	30	0.5316	255.43	5.11	1.15
4	7	10	21	0.4040	289.39	5.79	1.21
5	6 provided	N/A	32	0.5545	73.63	1.00	1.15
5	6	10	46	0.6695	292.35	3.97	1.15
5	6	20	36	0.5980	402.75	5.47	1.21
5	7	10	37	0.6020	324.19	4.40	1.21
5	7	15	27	0.4966	381.16	5.17	1.21

TABLE 5.5

3D elasticity problem, 201,720 degrees of freedom, with Young modulus featuring random jumps in $(1, 10^{\sigma})$.

tion of the candidate functions to an aggregate and subsequent local orthogonalization of these functions. It is therefore suitable for parallel processing as long as the aggregates are local to the processor. Parallelization of the underlying smoothed aggregation solver is currently under development. When it is completed, then αSA will also benefit from the parallel speedup. The parallel version is also expected to gain better parallel scalability by replacing the traditionally used Gauss-Seidel relaxation with the polynomial smoothing procedures investigated recently in [1]. The performance of the parallel implementation will depend on the quality of the parallel matrix-vector product.

Future development will concentrate on extending features of the underlying method on which αSA relies and on developing theory beyond the heuristics we developed here. Although most decisions are currently made by the code at runtime, much remains to be done to fully automate the procedure, such as determining certain tolerances that are now input by the user. We plan to explore the possibility of setting or updating these parameters at runtime based on the characteristics of the problem at hand. A related work in progress [7] explores adaptive ideas suitable in the context of the standard AMG method.

REFERENCES

[1] M. Adams, M. Brezina, and J. H. and Ray Tuminaro, *Parallel multigrid smoothing: Polynomial versus Gauss-Seidel*, Journal of Computational Physics, 188 (2003), pp. 593–610.

[2] J. Bramble, J. Pasciak, J. Wang, and J. Xu, *Convergence estimates for multigrid algorithm without regularity assumptions*, Math. Comp., 57 (1991).

[3] A. Brandt. A lecture given at CASC, Lawrence Livermore National Lab, June 2001.

[4] A. Brandt, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, 1984.

[5] A. Brandt and D. Ron, *Multigrid solvers and multilevel optimization strategies*. Manuscript, 2002.

[6] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, , and J. W. Ruge, *Algebraic multigrid based on element interpolation (AMGe)*, SIAM J. Sci. Comp., 22 (2000), pp. 1570–1592.

[7] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive algebraic multigrid (αAMG)*. In preparation, 2003.

[8] M. Brezina, C. I. Heberton, J. Mandel, and P. Vaněk, *An iterative method with convergence rate chosen a priori*, UCD/CCM Report 140, Center for Computational Mathematics, University of Colorado at Denver, February 1999. http://www-math.cudenver.edu/ccmreports/rep140.ps.gz.

[9] M. Brezina and P. Vaněk, *A black-box iterative solver based on a two-level Schwarz method*, Computing, 63 (1999), pp. 233–263. Dec 07, 1999.

[10] W. Briggs, V. Henson, and S. McCormick, *A Multigrid Tutorial, Second Edition*, SIAM, Philadelphia, 2000, 193 pps.

[11] T. Chartier, *Element-Based Alebraic Multigrid (AMGe) and Spectral AMGe*, PhD thesis, Univerity of Colorado at Boulder, Boulder, Colorado, 2001.

[12] T. Chartier, R. D. Falgout, V. E. Henson, J. Jones, T. Manteuffel, S. McCormick, J. Ruge, and P. S. Vassilevski, *Spectral AMGe (ρAMGe)*. Submitted to SISC., 2002.

[13] J. Fish and V. Belsky, *Generalized aggregation multilevel solver*, International Journal for Numerical Methods in Engineering, 40 (1997), pp. 4341–4361.

[14] S. McCormick and J. Ruge, *Algebraic multigrid methods applied to problems in computational structural mechanics*, in State of the Art Surveys on Comp. Mech., A. K. Noor and J. T. Oden, eds., ASME, New York, 1989. Part 2, Chapt. 1.

[15] J. Ruge, *Algebraic multigrid (AMG) for geodetic survey problems*, in Proc. Int'l Multigrid Conf., April 1983, Copper Mountain, CO, 1983.

[16] ———, *Final report on AMG02*, report, Gesellschaft fuer Mathematik und Datenverarbeitung, St. Augustin, 1985, 1985. GMD, Contract No. 5110/022090.

[17] ———, *Algebraic multigrid applied to systems of partial differential equations*, in Proc. Int'l Multigrid Conf., April 1985, S. McCormick, ed., North Holland, Amsterdam, Copper Mountain, CO, 1986.

[18] J. W. Ruge and K. Stüben, *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., The Institute of Mathematics and its Applications Conference Series, Clarendon Press, Oxford, 1985, pp. 169–212.

[19] ———, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1987, pp. 73–130.

[20] P. Vaněk, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.

[21] P. Vaněk, *Acceleration of convergence of a two-level algorithm by smoothing transfer operator*, Applications of Mathematics, 37 (1992), pp. 265–274.

[22] P. Vaněk, M. Brezina, and R. Tezaur, *Two-grid method for linear elasticity on unstructured meshes*, SIAM J. Sci. Comp., 21 (1999), pp. 900–923.

[23] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.