Title: Algebraic Multigrid Methods

Name: Luke Olson[1]

Affil./Addr.: Department of Computer Science

University of Illinois at Urbana-Champaign

Urbana, IL 61801

email: lukeo@illinois.edu

url: http://www.cs.uiuc.edu/homes/lukeo/

# Algebraic Multigrid Methods

## Synonyms

Algebraic Multigrid, AMG

## Definition

Algebraic multigrid (AMG) methods are used to approximate solutions to (sparse) linear systems of equations using the multilevel strategy of relaxation and coarse-grid correction that are used in *geometric* multigrid (GMG) methods. While partial differential equations (PDEs) are often the source of these linear systems, the goal in AMG is to generalize the multilevel process to target problems where the correct coarse problem is not apparent — e.g. unstructured meshes, graph problems, or structured problems where uniform refinement is not effective. In GMG, a multilevel hierarchy is determined from structured coarsening of the problem, followed by defining relaxation and interpolation operators. In contrast, in an AMG method the relaxation method is selected — e.g. Gauss-Seidel — and coarse problems and interpolation are automatically constructed.

# Overview

Early work in multigrid methods relied on geometric structure to construct coarse problems. This was generalized in [11] by McCormick, where multigrid is analyzed in terms of the matrix properties. This algebraic approach to theory was further extended by Mandal in [10], and together with [3] by Brandt, these works form the basis for much of the early development which led to the so-called Ruge-Stüben or *classical* algebraic multigrid (CAMG) method in [13].

One distinguishing aspect of CAMG is that the coarse problem is defined on a subset of the degrees-of-freedom of the problem, thus resulting in both coarse and fine points leading to the term CF-based AMG. A different style of algebraic multigrid emerged in [14] as *smoothed aggregation* based AMG (SA), where collections of degrees-of-freedom (or an aggregate) define a coarse degree-of-freedom. Together the frameworks of CF and SA based AMG have led to a number of developments in extending AMG to a wider class of problems and architectures.

There are a number of software libraries that implement different forms of AMG for different uses. The original CAMG algorithm and variants are available as `amg1r5` and `amg1r6` [13]. The Hypre library supports a parallel implementation of CF-based AMG in the BoomerAMG package [8]. The Trilinos package includes `ML` [7] as a parallel, SA-based AMG solver. Finally, PyAMG [2] includes a number of AMG variants for testings, and Cusp [1] distributes with a standard SA implementation for use on a graphics processing unit (GPU).

## Terminology

The goal of the AMG solver is to approximate the solution to

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

where $A \in \mathbb{R}^{n\times n}$ is sparse, symmetric, and positive definite. The *fine* problem (1) is defined on the fine index set $\Omega_0 = \{0, \ldots, n-1\}$. An AMG method is generally determined in two phases: the setup phase and the solve phase. The *setup* phase is responsible for constructing coarse operators $A_k$ for level $k$ of the hierarchy, along with interpolation operator $P_k$. A basic hierarchy, for example, consists of a series of operators $\{A_0, A_1, \ldots, A_m\}$ and $\{P_0, P_1, \ldots, P_{m-1}\}$.

Given such a hierarchy, the *solve* phase then executes in the same manner as that of geometric multigrid, as in Algorithm 1 for a *two*-level method; an $m$-level method extends similarly. Here, operator $\mathcal{G}(\cdot)$ denotes a relaxation method such as weighted Jacobi or Gauss-Seidel.

**Algorithm 1:** AMG Solve Phase

| | |
|---|---|
| $\mathbf{x} \leftarrow \mathcal{G}(A_0, \mathbf{x}, \mathbf{b})$ | {Pre-relaxation on $\Omega_0$} |
| $\mathbf{r}_1 \leftarrow P_0^T \mathbf{r}$ | {Restrict residual $\Omega_1$} |
| $\mathbf{e}_1 \leftarrow A_1^{-1} \mathbf{r}_1$ | {Coarse-grid solution on $\Omega_1$} |
| $\hat{\mathbf{e}} \leftarrow P_0 \mathbf{e}_1$ | {Interpolate coarse-grid error} |
| $\mathbf{x} \leftarrow \mathbf{x} + \hat{\mathbf{e}}$ | {Correct fine-grid solution} |
| $\mathbf{x} \leftarrow \mathcal{G}(A_0, \mathbf{x}, \mathbf{b})$ | {Post-relaxation on $\Omega_0$} |

## Theoretical Observations

The two grid process defined in Algorithm 1 can be viewed as an error propagation operator. First, let $G$ represent the error operator for relaxation — e.g. $G = I - \omega D^{-1} A$ for weighted Jacobi. In addition, coarse operators $A_k$ are typically defined through a Galerkin product: $A_{k+1} = P_k^T A_k P_k$. Thus for an initial guess $x$ and exact solution $x^*$ to (1), the error $e = x^* - x$ for a two-grid method with one pass of pre-relaxation is defined through

$$e \leftarrow \Big(I - P_0(P_0^T A_0 P_0)^{-1} P_0^T A_0\Big) Ge \tag{2}$$

correct: $I$ — interpolate: $P_0$ — coarse solve: $(P_0^T A_0 P_0)^{-1}$ — restrict: $P_0^T$ — residual: $A_0$ — relax: $G$

A key observation follows from (2) in defining AMG methods: if the error remaining after relaxation is contained in the range of interpolation, denoted $\mathcal{R}(P)$, then the solver is exact. That is, if $Ge \in \mathcal{R}(P)$, then coarse grid correction defined by $T = I - P(P^T AP)^{-1} P^T A$ annihilates the error. One important property of $T$ is that it is an $A$-orthogonal projection, which highlights the close relationship with other subspace projection methods.

# Methods

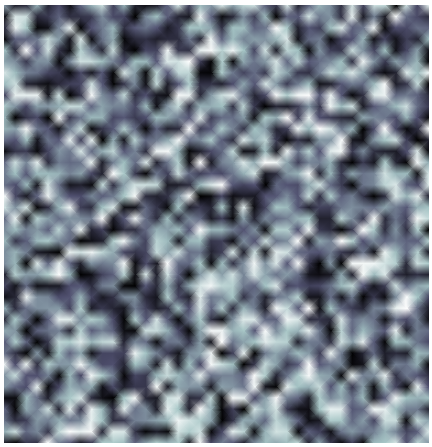
The setup phase of AMG defines the method, however there are several common features

1. Determining the strength of connection between degrees of freedom;
2. Identifying coarse degrees of freedom;
3. Constructing interpolation, $P$; and
4. Forming the coarse operator through the Galerkin product $P^T AP$.

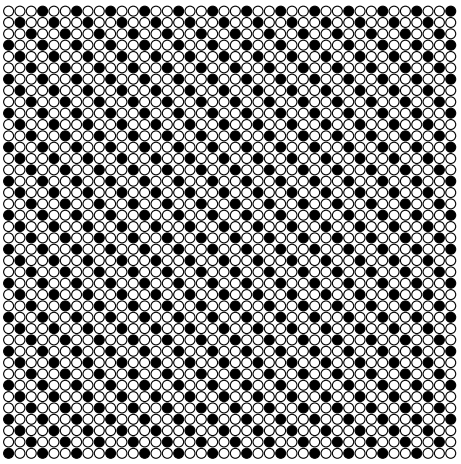
Algebraic methods determine coarse grids and the resulting interpolation operators to *complement* the limitations of relaxation. That is, interpolation should capture the error components that relaxation, e.g. weighted Jacobi, does not adequately reduce. The error not reduced by relaxation is termed *algebraically smooth* error. To identify smooth error, an edge in the graph of matrix $A$ is deemed *strong* if error is perceived to vary slowly along that edge. This allows for automatic coarsening to match the behavior of relaxation.

As an example, consider the case of an anisotropic diffusion operator $-u_{xx} - \varepsilon u_{yy}$ rotated by $45^\circ$ along the coordinate axis and discretized by Q1 finite elements on

a uniform mesh. As the anisotropic behavior increases ($\varepsilon \to 0$), uniform coarsening with geometric multigrid results in degraded performance. In an algebraic method, coarsening is along the direction of smooth error, which follows the line of anisotropy as shown in Figure 1. Here, coarsening is only performed (automatically) in the direction of smooth error and results in high convergence.



(a) Error after relaxation for a random guess.



(b) Coarse points (•) and fine points (○).

Fig. 1: CF-based AMG for a rotated anisotropic diffusion problem.

## CF-based AMG

CF-based AMG begins with $A_k$, the $k$-level matrix, and determines strong edges according to

$$-A_{ij} \geq \theta \max_{k \neq i} -A_{ik}, \tag{3}$$

where $\theta$ is some threshold. This process yields a strength matrix $S$ (see Algorithm 2), which identifies edges where error is smooth after relaxation. In turn, $S$ is used to split the index set into either $C$-points or $F$-points (see Figure 1b), requiring that $F$ points are strongly connected to at least one $C$-point (for interpolation). With $C/F$-points identified, weights $W$ are determined to form an interpolation operator of the form

$$P = \begin{bmatrix} W \\ I \end{bmatrix}$$

Finally a coarse operator is constructed through a Galerkin product, $P^T AP$, which is the dominant cost for most AMG methods.

**Algorithm 2:** CF-based AMG

**Input**: $A$: $n \times n$ fine level matrix

**Return**: $A_0, \ldots, A_m$,$P_0, \ldots, P_{m-1}$

**for** $k = 0, \ldots, m-1$ **do**

| | |
|---|---|
| $S \leftarrow \texttt{strength}(A_k, \theta)$ | {Compute strength-of-connection} |
| $C$, $F \leftarrow \texttt{split}(S)$ | {Determine $C$-points and $F$-points} |
| $P_k \leftarrow \texttt{interp}(A_k, C, F)$ | {Construct interpolation from $C$ to $F$} |
| $A_{k+1} = P_k^T A_k P_k$ | {Construct coarse operator} |

## SA-based AMG

SA-based AMG methods have an important distinction: they require *a priori* knowledge of the slow-to-converge or smooth error, denoted $B$. A common choice for these vectors in the absence of more knowledge about the problem is $B \equiv \mathbf{1}$, the constant vector. The SA algorithm (see Algorithm 3) first constructs a strength-of-connection matrix, similar to CF-based AMG, but using the symmetric threshold

$$|A_{ij}| \geq \theta \sqrt{|A_{ii} A_{jj}|}. \tag{4}$$

From this, aggregates or collections of nodes are formed (see Figure 2) and represent coarse degrees of freedom. Next, $B$ is restricted locally to each aggregate to form a *tentative* interpolation operator $T$ so that $B \in \mathcal{R}(T)$. Then, to improve the accuracy of interpolation, $T$ is smoothed (for example with weighted Jacobi) to yield interpolation operator $P$. This is shown in Figure 2b where piecewise constant functions form the

basis for the range of $T$, while the basis for the range of $P$ resembles piecewise linear functions. Finally, the coarse operator is computed through the Galerkin product.

**Algorithm 3:** SA-based AMG

**Input**: $A$: $n \times n$ fine level matrix

$B$: $n \times c$ vectors representing $c$ smooth error components

**Return**: $A_0, \ldots, A_m, P_0, \ldots, P_{m-1}$

**for** $k = 0, \ldots, m-1$ **do**

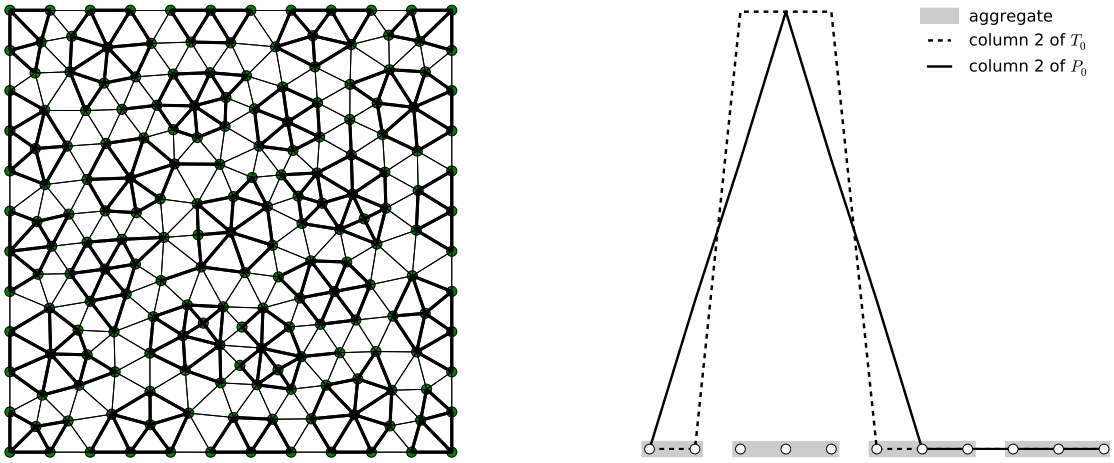
$S \leftarrow \texttt{strength}(A_k, \theta)$ {Compute strength-of-connection}

$Agg \leftarrow \texttt{aggregate}(S)$ {Aggregate nodes in the strength graph}

$T_k \leftarrow \texttt{tentative}(B, Agg)$ {Construct tentative interpolation operator}

$P_k \leftarrow \texttt{smooth}(A_k, T_k)$ {Improve interpolation operator}

$A_{k+1} = P_k^T A_k P_k$ {Construct coarse operator}




(a) Aggregation of nodes on a mesh.

(b) Column of $T$ and $P$ on an aggregate.

Fig. 2: SA-based AMG in 2D and in 1D.

## Practical Considerations

Algebraic multigrid methods are commonly used as preconditioners — for example to restarted GMRES or conjugate gradient Krylov methods — leading to a reduction in the number of iterations. However, the total cost of the preconditioned iteration requires

an assessment of both the convergence factor $\rho$ and the work in each multigrid cycle. To measure the work in a V-cycle the so-called *operator complexity* of the hierarchy is used: $c_{\text{op}} = \frac{\sum_{k=0}^{m} \texttt{nnz}(A_k)}{\texttt{nnz}(A_0)}$. With this, the total *work per digit of accuracy* is estimated as $c_{\text{op}}/\log_{10}\rho$. This relates the cost of an AMG cycle to the cost of a standard sparse matrix-vector multiplication. This also exposes the cost versus accuracy relationship in AMG, yet this may be "hidden" if the cost of the setup phase is not included.

In both CF-based AMG and SA-based AMG, the interpolation operator plays a large role in both the effectiveness and the complexity of the algorithm. In each case, interpolation can be enriched — for example by extending the interpolation pattern or by growing $B$ in the case of SA — leading to faster convergence, but more costly iterations.

There are a number of ways in which AMG has been extended or redesigned in order to increase the robustness for a wider range of problems or to improve efficiency. For example, the adaptive methods of [4; 5] attempt to construct an improved hierarchy by modifying the setup phase based on its performance on $A\mathbf{x} = 0$. Other works focus on individual components, such as generalizing strength of connection [12] or coarsening, such as the work of *compatible relaxation* [9], where coarse grids are selected directly through relaxation. And new methods continue to emerge as the theory supporting AMG becomes more developed and generalized [6].

## Cross-references

iterative methods, Krylov methods, preconditioning, multigrid, domain decomposition

## References

1. Bell N, Garland M (2012) Cusp: Generic parallel algorithms for sparse matrix and graph computations. URL http://cusp-library.googlecode.com, version 0.3.0

2. Bell WN, Olson LN, Schroder JB (2011) PyAMG: Algebraic multigrid solvers in Python v2.0. URL http://www.pyamg.org, release 2.0
3. Brandt A (1986) Algebraic multigrid theory: The symmetric case. Appl Math Comput 19:23–56
4. Brezina M, Falgout R, MacLachlan S, Manteuffel T, McCormick S, Ruge J (2004) Adaptive smoothed aggregation ($\alpha$sa). SIAM Journal on Scientific Computing 25(6):1896–1920
5. Brezina M, Falgout R, MacLachlan S, Manteuffel T, McCormick S, Ruge J (2006) Adaptive algebraic multigrid. SIAM Journal on Scientific Computing 27(4):1261–1286
6. Falgout R, Vassilevski P (2004) On generalizing the algebraic multigrid framework. SIAM Journal on Numerical Analysis 42(4):1669–1693
7. Gee MW, Siefert CM, Hu JJ, Tuminaro RS, Sala MG (2007) ML 5.0 Smoothed Aggregation Users Guide. URL http://trilinos.org/packages/ml/
8. Henson VE, Yang UM (2002) Boomeramg: A parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics 41(1):155 – 177
9. Livne OE (2004) Coarsening by compatible relaxation. Numerical Linear Algebra with Applications 11(2-3):205–227
10. Mandel J (1988) Algebraic study of multigrid methods for symmetric, definite problems. Appl Math Comput 25(1, part I):39–56
11. McCormick SF (1985) Multigrid methods for variational problems: general theory for the $V$-cycle. SIAM J Numer Anal 22(4):634–643
12. Olson LN, Schroder J, Tuminaro RS (2010) A new perspective on strength measures in algebraic multigrid. Numerical Linear Algebra with Applications 17(4):713–733
13. Ruge JW, Stüben K (1987) Algebraic multigrid. In: Multigrid methods, Frontiers Appl. Math., vol 3, SIAM, Philadelphia, PA, pp 73–130
14. Vaněk P, Mandel J, Brezina M (1996) Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing 56(3):179–196