Progressive Multigrid Eigensolvers for Multiscale Spectral Segmentation

Michael Maire[1] and Stella X. Yu[2]
[1]California Institute of Technology - Pasadena, CA
[2]University of California at Berkeley / ICSI - Berkeley, CA
mmaire@caltech.edu, stellayu@berkeley.edu

Abstract

We reexamine the role of multiscale cues in image segmentation using an architecture that constructs a globally coherent scale-space output representation. This characteristic is in contrast to many existing works on bottom-up segmentation, which prematurely compress information into a single scale. The architecture is a standard extension of Normalized Cuts from an image plane to an image pyramid, with cross-scale constraints enforcing consistency in the solution while allowing emergence of coarse-to-fine detail.

We observe that multiscale processing, in addition to improving segmentation quality, offers a route by which to speed computation. We make a significant algorithmic advance in the form of a custom multigrid eigensolver for constrained Angular Embedding problems possessing coarse-to-fine structure. Multiscale Normalized Cuts is a special case. Our solver builds atop recent results on randomized matrix approximation, using a novel interpolation operation to mold its computational strategy according to cross-scale constraints in the problem definition. Applying our solver to multiscale segmentation problems demonstrates speedup by more than an order of magnitude. This speedup is at the algorithmic level and carries over to any implementation target.

1. Introduction

Spectral clustering techniques have wide applicability to perceptual organization problems. The spectral relaxation of the Normalized Cuts problem [14] appears as the driving method in much research on image segmentation [5, 13, 17, 15, 1, 11]. The importance of exploiting multiscale cues to generate high quality segmentations is widely recognized. Yu [15] formulates segmentation as clustering on the average of multiscale affinity matrices. Arbeláez *et al*. [1] derive entries of a single affinity matrix from a combination of multiscale features. Both of these systems summarize multiscale cues and optimize a single output that best explains the summary.

Cour *et al*. [5] present an alternative, suggesting multirange or multiscale approaches which couple sparse affinity matrices at coarse and fine levels of an image pyramid using constraints [17]. Though this work lacks the sophisticated post-processing steps found in other spectral segmentation pipelines, such as gPb [1], it offers the insight that scale-space representation should be preserved throughout the clustering procedure.

While image pyramids offer concise means of describing both short and long range interactions, they do not alleviate the fact that the computation time required to solve spectral clustering problems is often prohibitive. Both image pyramids and the Nyström method [6] can be seen as techniques for approximating dense affinity matrices. But, runtime for Normalized Cuts with only sparse affinity matrices on a modern CPU can still be measured in minutes [5, 1].

Sparse affinity matrices make these problems feasible in terms of asymptotic computational complexity, but far from fast. Multigrid methods [13, 4, 2, 8] provide a strategy for further reducing the computational load.

Our key observation is that multigrid and multiscale techniques are complimentary and should be intertwined, producing fast high-quality segmentation algorithms. Unlike generic multigrid methods [4, 2, 8], we explicitly address constrained problems [17, 5] with the intuition that the constraints themselves can guide the schedule of computations within the solver. Figure 1 provides a comparison.

Sections 2 and 3 present eigensolver technical details in the more general setting of Angular Embedding, an extension of Normalized Cuts to handle both grouping and ordering relationships [16], recently used in simultaneously resolving segmentation, figure-ground, and object detection [9, 10]. We develop our solver within the framework of randomized matrix approximation [7], a new mathematical technique which naturally fits our interpolation strategy. We also inherit its favorable parallelization properties.

Section 4 demonstrates speedup results as well as segmentation improvements, in the form of high quality contours, achieved by combining our efficient solver with the best aspects of previous systems [5, 1]. Section 5 concludes.

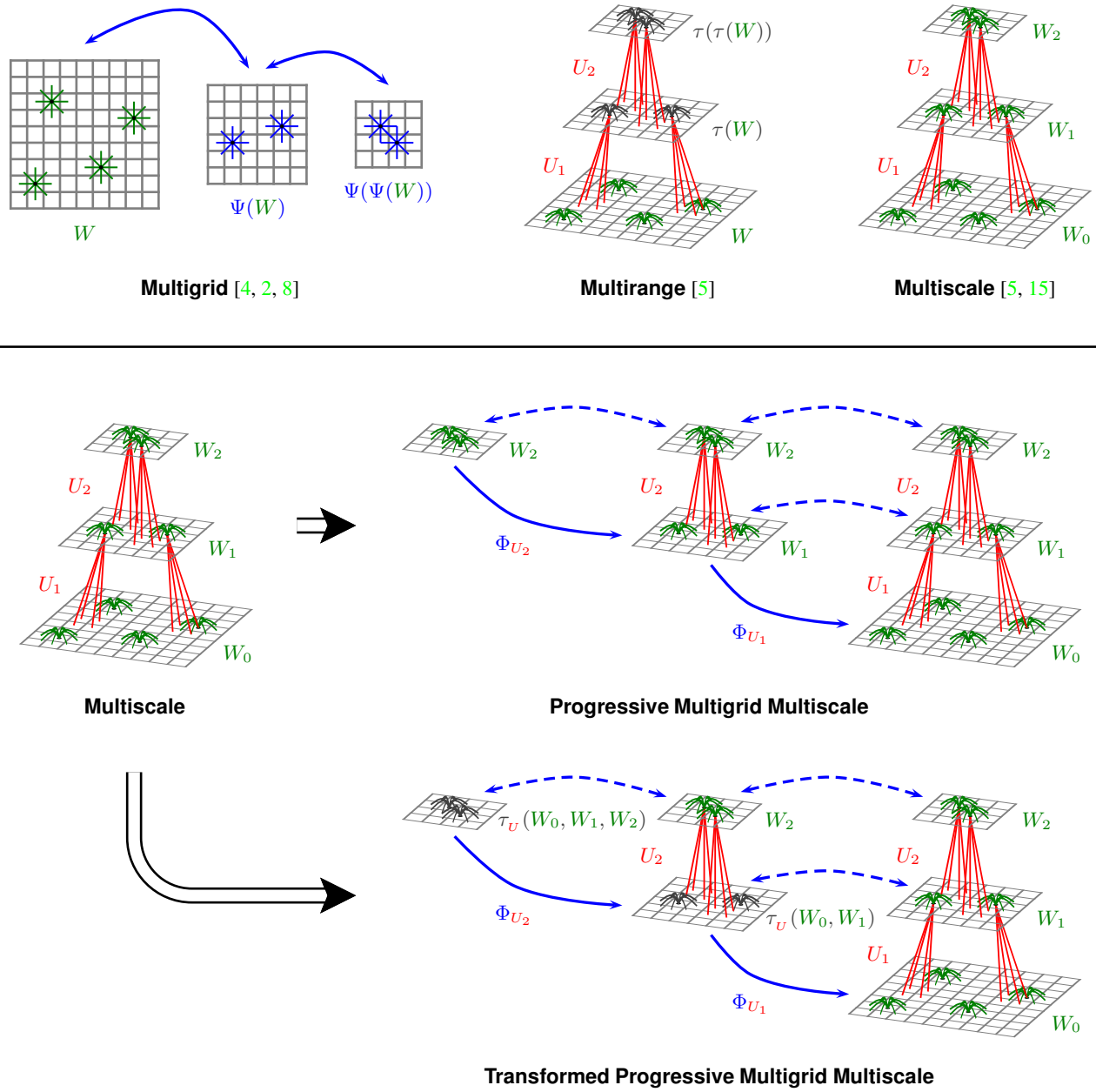

Figure 1. **System comparison.** Multigrid techniques exploit coarse-to-fine structure within a spectral clustering problem by adapting the optimization routine used to solve it. Multirange or multiscale techniques instead adapt the problem definition to explicitly encode such structure. We combine both approaches, producing a progressive multigrid algorithm for the solver. *Top Left:* Consider a sparse matrix W defining pairwise affinities between nodes in a graph (*e.g.* connections between neighboring pixels, for image segmentation, as shown in green). Generic multigrid eigensolvers [4, 2, 8], applied to the corresponding Normalized Cuts eigenproblem [14], coarsen the problem by subsampling nodes and interpolating weights. Solution eigenvectors from iteratively coarsened problems, $\Psi(W)$ and $\Psi(\Psi(W))$, initialize the solver on the next finer problems, W and $\Psi(W)$, respectively (blue arrows). *Top Middle:* Multirange [5] simulates the effect of a dense affinity matrix by sampling longer-range affinities on coarser pixel grids ($\tau(W)$) and tying graphs together using constraints (U_s, red links) [17]. *Top Right:* Rather than resampling, a true multiscale approach [5, 15] ties together level-dependent information, in the form of different affinities, W_0, W_1, W_2, on each subgraph. *Bottom:* Our custom eigensolver maps a multiscale constrained spectral clustering problem onto a progressive multigrid computation strategy. Unlike generic multigrid methods, the constraints from the problem definition shape computation within the solver. Instead of coarsening uniformly, our algorithm drops or adds entire levels at once. Constraints both tie levels in the expanded problem and determine interpolation functions Φ_{U_s} for moving work between levels. When dropping a level, say W_0, we can optionally use the constraints to fold it into the next coarsest level, substituting transformed affinity $\tau_U(W_0, W_1)$ for W_1.

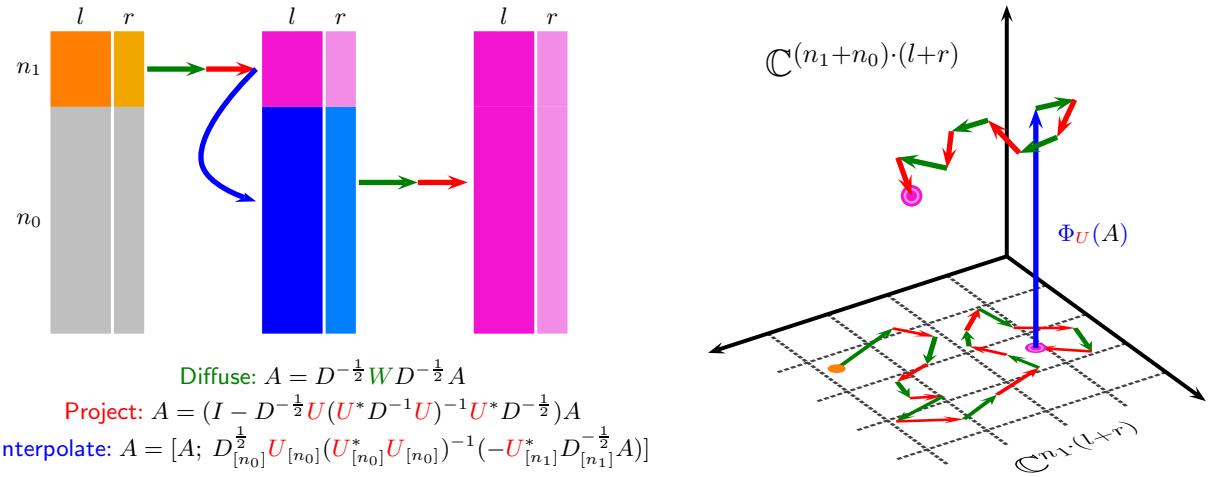

Figure 2. **Multiscale evolution from random noise to eigenvectors.** *Left:* Let there be n_1 and n_0 nodes at coarse and fine scales respectively, with $\tilde{n}_0 = n_0 + n_1$ total nodes. To compute m length $\tilde{n}_0$ eigenvectors, within matrix A we store $l = 2m$ random vectors (l-block) for spanning the eigenspace, and $r = m$ more vectors (r-block) for testing convergence. We build these vectors progressively over scale, at each step applying diffusion and projection based on the graph weights and constraints, and then checking whether the r vectors lie in the l space. We first initialize (orange) the coarse scale $n_1 \times (l + r)$ vectors (top block) with random Gaussian noise, and follow with diffusion and projection, repeating until convergence (pink). We then use the top block to initialize the (bottom-block) fine-scale $n_0 \times (l+r)$ vectors via interpolation (blue) defined by inter-scale constraints. This is followed again by diffusion, projection, and checking the entire matrix A. Upon convergence, the r block is no longer of any use. We apply diffusion to the l block before collapsing it to a core $l \times l$ matrix B. We extract m eigenvectors of this much smaller matrix B and then interpolate back to recover the desired m eigenvectors of length $\tilde{n}_0$. *Right:* In an equivalent view, A initially lives in a subspace of dimension $n_1(l + r)$, where diffusion and projection operations are cheap. Performing most of the work in this subspace before interpolating to deal with the full problem in the larger space gives us a speedup.

2. Spectral Clustering with Constraints

We consider Angular Embedding (AE) [16] problems with constraints [17], defined by a triple (C, Θ, U) of real-valued matrices. Skew-symmetric $n \times n$ matrix Θ specifies relative ordering relationships between n nodes. Symmetric $n \times n$ matrix C specifies a confidence on each relationship. Normalized Cuts is a special case, where $\Theta = 0$ and confidence is synonymous with affinity. The task is to embed nodes into an m-dimensional space, such that location in this embedding space preserves the pairwise relationships.

The $n \times u$ matrix U specifies u linear constraints that the solution embedding x must satisfy: $U^* x = 0$, where * denotes complex conjugate transpose. In the case of multiscale segmentation, U will state that each coarse pixel must be consistent with the finer pixels in the scale below it; the coarse pixel's embedding must be the average of the fine.

The optimal embedding is given by the leading m eigenvectors of the generalized eigenproblem:

$$QPQx = \lambda x \tag{1}$$

where P is a normalized weight matrix and Q is a projector onto the feasible solution space (Q enforces constraints):

$$P = D^{-1} W \tag{2}$$

$$Q = I - D^{-1} U (U^T D^{-1} U)^{-1} U^T \tag{3}$$

with D and W defined in terms of C and Θ by:

$$D = \mathrm{diag}(C1) \qquad W = C \bullet \exp(i\Theta) \tag{4}$$

where 1 is a column vector of n ones, I is the identity matrix, $\mathrm{diag}(\cdot)$ is a matrix with its vector argument on the main diagonal, $\bullet$ denotes the matrix Hadamard product, $i = \sqrt{-1}$ and exponentiation acts element-wise. For convenience, we work with degree-normalized variable $z = D^{\frac{1}{2}} x$ with correspondingly modified $\bar{P}$, $\bar{Q}$, and $\bar{U}$ replacing P, Q, U.

The multiscale setting upgrades each of C, Θ, U to an array of matrices, $\mathbf{C}$, $\mathbf{\Theta}$, $\mathbf{U}$, indexed by level s. Let n_s denote the number of nodes at level s and $\tilde{n}_s = \sum_{\acute{s} \geq s} n_{\acute{s}}$ the number of nodes in levels s and coarser.

Node relationships are within-level only, making $\mathbf{C}_s$, $\mathbf{\Theta}_s$ $n_s \times n_s$ matrices. Constraints must appear incrementally, associating nodes newly appearing at level s with nodes from coarser levels. Hence, $\mathbf{U}_s$ has dimensions $\tilde{n}_s \times u_s$.

3. Eigensolver

Let $M_s = Q_s P_s Q_s$ denote the matrix whose leading eigenvectors solve the multiscale AE problem $(\mathbf{C}, \mathbf{\Theta}, \mathbf{U})$ restricted to levels s and coarser. The intuition behind our eigensolver is to interpolate from the eigenvectors of M_s an initial solver state for M_{s-1}, eventually obtaining the eigenvectors of M_0 and thereby solving the unrestricted problem. Coarser subproblems speed the solution to finer ones.

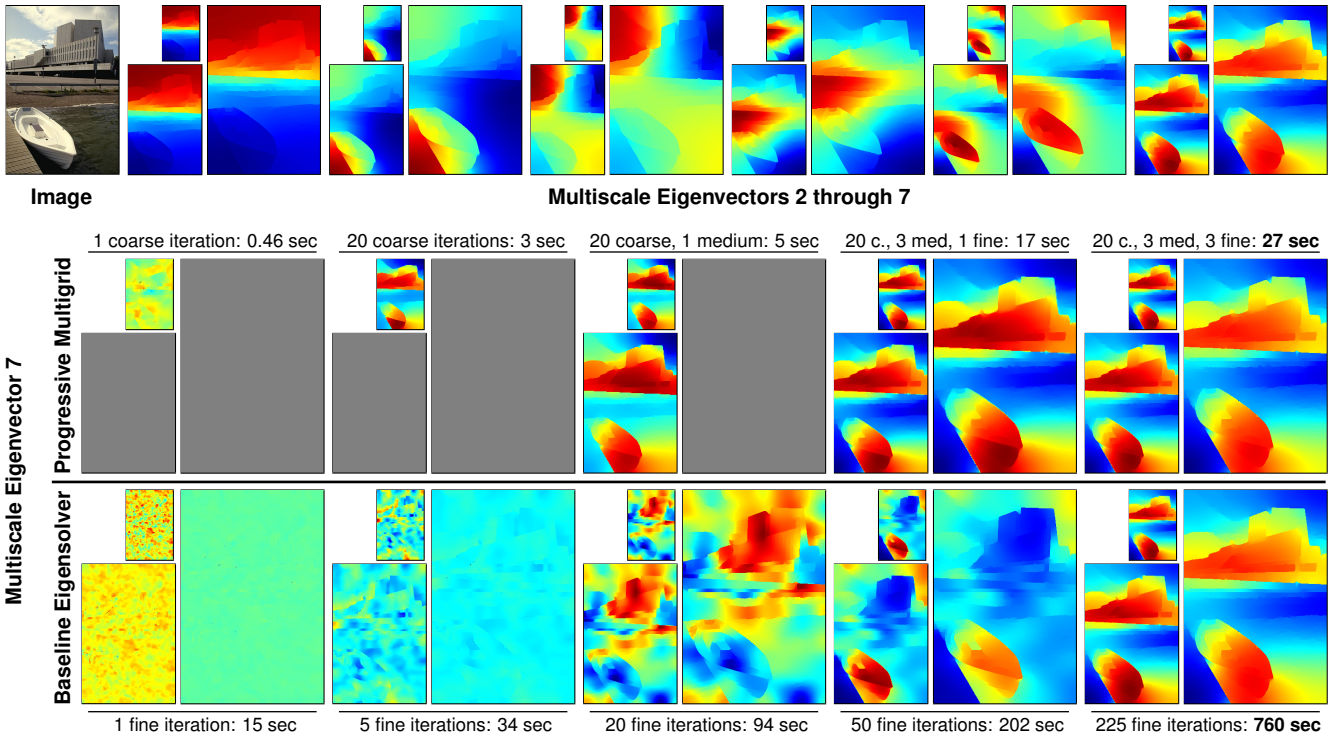

Figure 3. **Eigenvector convergence comparison.** *Top:* Image and leading eigenvectors for multiscale Normalized Cuts applied across a three-level image pyramid with scales linked by constraints. *Bottom:* Our progressive multigrid solver processes sub-pyramids in coarse-to-fine order. The baseline solver immediately starts work on the finest pyramid, taking far longer to converge (760 sec vs 27 sec).

To accomplish this, we borrow the randomized subspace iteration procedure from recent results concerning probabilistic algorithms for constructing matrix decompositions [7]. Instead of working directly with large sparse matrices M_s and their eigenvectors, we incrementally construct a tall dense matrix A whose range approximates the range of M_0. Computation of approximation A for M proceeds by sampling a sufficient number of random vectors and repeatedly applying M until these vectors form a basis A that captures the range of M. Though seemingly similar to a power iteration method for finding eigenvectors, we do not yet extract them. A itself is not a set of eigenvectors, but later they can be cheaply obtained from A. When desiring m eigenvectors, we must oversample the size of the basis A (sampling $2m$ vectors is sufficient) in order the ensure the randomized algorithm has a negligible (exponentially small) probability of failure.

We add a novel interpolation step to randomized subspace iteration in order to initialize A_{s-1} from A_s. As coarse and fine subproblems share the coarse levels, initialization is a copy operation on these coarse levels and an interpolation for the finest level (see dotted and solid blue arrows in Figure 1). We equivalently work with a single matrix A and grow it by adding rows during interpolation.

Suppose we have a two-level problem with cross-scale constraint $U^*x = 0$. This can be rewritten as:

$$\left[U_{[n_1]};\ U_{[n_0]}\right]^* \left[x_{[n_1]};\ x_{[n_0]}\right] = 0 \tag{5}$$

where $U_{[n_1]}$ is the $n_1 \times u$ upper block of U involving values for the n_1 coarse nodes and $U_{[n_0]}$ is the lower block with values for the n_0 fine nodes. The notation similarly selects subranges of x. Given only x_1, solving this underconstrained equation for x_0 in the least squares sense allows us to interpolate a fine representation from a coarse one. We apply precisely the same interpolation procedure during coarse-to-fine subspace iteration, with x replaced by the appropriate subblock of A.

Changing variables from x to z, Figure 2 illustrates the core operations within our eigensolver. Determining convergence requires evolving two separate bases within A and checking the accuracy with which the first reconstructs the second. Algorithms 1 and 2 present full technical details.

Algorithm 2's outer loop iterates over coarse-to-fine pyramids. Lines 4-16 extract the active subproblem; here $\mathrm{Diag}(\cdot)$ places its matrix arguments on the block diagonal of a larger matrix. Lines 17-27 initialize A or interpolate from a coarser level. Lines 28-33 perform almost all computational work, refining A until the solution converges for the subproblem; here $k \leftarrow 2k$ guarantees that asymptotically we waste negligible time convergence testing. Lines

Algorithm 1 Matrix approximation via subspace iteration

Given functions f, g such that $f(X) = MX$ and $g(X) = M^*X$ for some $n \times n$ matrix M, compute an $n \times (l + r)$ matrix A whose leftmost l columns approximate the range of M and rightmost r columns test convergence [7].

Initialize A using Gaussian random sampling.

function MXAPPROXINIT(f, n, l, r)
  draw $n \times (l + r)$ Gaussian matrix $\Omega \in \mathbb{C}^{n\cdot(l+r)}$
  $A \leftarrow$ MXAPPROXREORTH($f(\Omega), l, r$)
return A

Perform a single update to A to improve the approximation.

function MXAPPROXUPDATE(f, g, A, l, r)
  $A \leftarrow g(A)$
  $A \leftarrow$ MXAPPROXREORTH(A, l, r) ▷ *optional*[1]
  $A \leftarrow f(A)$
  $A \leftarrow$ MXAPPROXREORTH(A, l, r) ▷ *optional*[1]
return A

Perform k updates to A to improve the approximation. Also return $\hat{\hat{A}}$, the left l columns of A just before the final update.

function MXAPPROXREFINE(f, g, A, l, r, k)
  for $j \leftarrow 0, \ldots, (k-2)$ **do**
    $A \leftarrow$ MXAPPROXUPDATE(f, g, A, l, r)
  end for
  $(\hat{\hat{A}}, \_) \leftarrow$ MXAPPROXSPLIT(A, l, r)
  $\hat{\hat{A}} \leftarrow$ QR-ORTHONORMALIZE($\hat{\hat{A}}$)
  $A \leftarrow$ MXAPPROXUPDATE(f, g, A, l, r) ▷ *final one*
  $A \leftarrow$ MXAPPROXREORTH(A, l, r)
return $(A, \hat{\hat{A}})$

Reorthonormalize bases in the left/rightmost columns of A.

function MXAPPROXREORTH(A, l, r)
  $(\widehat{A}, \breve{A}) \leftarrow$ MXAPPROXSPLIT(A, l, r)
  $\widehat{A} \leftarrow$ QR-ORTHONORMALIZE($\widehat{A}$) ▷ *left basis*
  $\breve{A} \leftarrow$ QR-ORTHONORMALIZE($\breve{A}$) ▷ *right basis*
return $\left[\widehat{A}\ \ \breve{A}\right]$ ▷ *recombine A*

Split A into the bases in its left/rightmost columns.

function MXAPPROXSPLIT(A, l, r)
  $\widehat{A} \leftarrow A_{[0:(n-1),\ 0:(l-1)]}$ ▷ *leftmost l columns*
  $\breve{A} \leftarrow A_{[0:(n-1),\ l:(l+r-1)]}$ ▷ *rightmost r columns*
return $(\widehat{A}, \breve{A})$

Test for convergence by returning an error bound estimate.

function MXAPPROXTEST($\hat{\hat{A}}, \breve{A}$)
  $E \leftarrow \breve{A} - \hat{\hat{A}}\hat{\hat{A}}^*\breve{A}$ ▷ *$n \times r$ error matrix*
return $\max_{j=0,\ldots,r-1} \left\| E_{[0:(n-1),\ j]} \right\|$

[1] Reorthonormalization here guarantees numerical stability. In practice, these calls can be executed rarely; we found no issue dropping them.

Algorithm 2 Progressive multigrid Angular Embedding

Compute the m leading eigenvectors V and eigenvalues Λ of a multiscale constrained Angular Embedding problem.

function MULTIGRIDAE($\mathbf{C}, \mathbf{\Theta}, \mathbf{U}, m$)
  $A \leftarrow [\,],\ \ l \leftarrow 2m,\ \ r \leftarrow m$
  for $s \leftarrow s_{\max}, \ldots, 0$ **do** ▷ *loop over scales*
    $C \leftarrow \mathrm{Diag}(\mathbf{C}_{s_{\max}}, \ldots, \mathbf{C}_s)$ ▷ *setup subproblem*
    $\Theta \leftarrow \mathrm{Diag}(\mathbf{\Theta}_{s_{\max}}, \ldots, \mathbf{\Theta}_s)$
    $D \leftarrow \mathrm{diag}(C1)$
    $W \leftarrow C \bullet \exp(i\Theta)$
    $\bar{P} \leftarrow D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$
    $U \leftarrow [\mathbf{U}_{s_{\max}}; \ldots; \mathbf{U}_s]$
    if $U = [\,]$ **then** ▷ *no constraints*
      $f(\cdot) \leftarrow$ DIFFUSE($\bar{P}, \cdot$)
    else ▷ *constraints*
      $\bar{U} \leftarrow D^{-\frac{1}{2}} U$
      $\bar{R} \leftarrow$ INCCHOLESKY($\bar{U}^*\bar{U}$)
      $f(\cdot) \leftarrow$ DIFFUSEPROJECT($\bar{P}, \bar{U}, \bar{R}, \cdot$)
    end if
    if $A = [\,]$ **then** ▷ *initialize*
      $A \leftarrow$ MXAPPROXINIT($f, \tilde{n}_s, l, r$)
    else ▷ *interpolate*
      $\hat{U}_\alpha \leftarrow U_{[0:(\tilde{n}_{s+1}-1),\ (\tilde{u}_{s+1}):(\tilde{u}_s-1)]}$
      $\hat{U}_\beta \leftarrow U_{[(\tilde{n}_{s+1}):(\tilde{n}_s-1),\ (\tilde{u}_{s+1}):(\tilde{u}_s-1)]}$
      $\hat{R}_\beta \leftarrow$ INCCHOLESKY($\hat{U}_\beta^*\hat{U}_\beta$)
      $\hat{D}_\alpha \leftarrow D_{[0:(\tilde{n}_{s+1}-1),\ 0:(\tilde{n}_{s+1}-1)]}$
      $\hat{D}_\beta \leftarrow D_{[(\tilde{n}_{s+1}):(\tilde{n}_s-1),\ (\tilde{n}_{s+1}):(\tilde{n}_s-1)]}$
      $\bar{\hat{U}}_\alpha \leftarrow \hat{D}_\alpha^{-\frac{1}{2}}\hat{U}_\alpha, \quad \bar{\hat{U}}_\beta \leftarrow \hat{D}_\beta^{\frac{1}{2}}\hat{U}_\beta$
      $A \leftarrow \left[A;\ \bar{\hat{U}}_\beta(\hat{R}_\beta \setminus \hat{R}_\beta^* \setminus (-\bar{\hat{U}}_\alpha^* A))\right]$
    end if
    $k \leftarrow 1$
    repeat ▷ *apply diffusion/projection*
      $(A, \hat{\hat{A}}) \leftarrow$ MXAPPROXREFINE(f, f, A, l, r, k)
      $(\ \_, \breve{A}) \leftarrow$ MXAPPROXSPLIT(A, l, r)
      $k \leftarrow 2k$
    until (MXAPPROXTEST($\hat{\hat{A}}, \breve{A}$) $< \epsilon$)
  end for
  $(\hat{A}, \_) \leftarrow$ MXAPPROXSPLIT(A, l, r)
  $B \leftarrow \hat{A}^* f(\hat{A})$ ▷ *B is an $l \times l$ matrix*
  $(V, \Lambda) \leftarrow$ EIGS(B, m) ▷ *small eigenproblem*
  $V \leftarrow D^{-\frac{1}{2}}\hat{A}V$ ▷ *$\tilde{n}_0 \times m$ eigenvectors*
return (V, Λ)

Apply eigensolver diffusion and projection operations.

function DIFFUSE($\bar{P}, Z$) **return** $\bar{P}Z$
function DIFFUSEPROJECT($\bar{P}, \bar{U}, \bar{R}, Z$)
  $Z \leftarrow Z - \bar{U}(\bar{R} \setminus \bar{R}^* \setminus \bar{U}^* Z)$
  $Z \leftarrow \bar{P}Z$
  $Z \leftarrow Z - \bar{U}(\bar{R} \setminus \bar{R}^* \setminus \bar{U}^* Z)$
return Z

35-38 solve a trivially small eigenproblem and recover the solution to the original Angular Embedding problem.

An important point is that we never explicitly construct $\overline{M} = \bar{Q}\bar{P}\bar{Q}$ as it may become dense even though $\bar{P}$ and $\bar{Q}$ are sparse. Yu and Shi [17] discuss options for resolving this issue. We adopt the one of using the incomplete Cholesky factorization of $(\bar{U}^*\bar{U})$ and solving a linear system. Notation $R \setminus z$ in the pseudocode means to solve $Ry = z$ and return y. Using the same trick, we avoid computing the explicit inverse $(\widehat{U}_\beta^*\widehat{U}_\beta)^{-1}$ when interpolating.

Algorithm 3 implements weight folding for the transformed system show in Figure 1, optionally replacing lines 6-7 of Algorithm 2 or, more efficiently, being conducted in an initial pass. For segmentation, we do not see significant differences with weight folding, so report results without.

Algorithm 3 Weight folding for transformed multigrid AE

Compute degree matrix D and weight matrix W that are active for the pyramid based at level s when solving the multiscale AE problem $(\mathbf{C}, \mathbf{\Theta}, \mathbf{U})$ using transformed multigrid.

1: **function** WEIGHTFOLD($\mathbf{C}, \mathbf{\Theta}, \mathbf{U}, s$)
2: $C \leftarrow \mathrm{Diag}(\mathbf{C}_{s_{\max}}, \ldots, \mathbf{C}_0)$ ▷ *initialize weights*
3: $\Theta \leftarrow \mathrm{Diag}(\mathbf{\Theta}_{s_{\max}}, \ldots, \mathbf{\Theta}_0)$
4: $D \leftarrow \mathrm{diag}(C1)$
5: $W \leftarrow C \bullet \exp(i\Theta)$ ▷ $\tilde{n}_0 \times \tilde{n}_0$ *matrix*
6: $U \leftarrow [\mathbf{U}_{s_{\max}}; \ldots; \mathbf{U}_0]$
7: **for** $\acute{s} \leftarrow 0, \ldots, (s-1)$ **do** ▷ *fold levels below* s
8: $\widehat{U}_\alpha \leftarrow U_{[0:(\tilde{n}_{\acute{s}+1}-1),\ (\tilde{u}_{\acute{s}+1}):(\tilde{u}_{\acute{s}}-1)]}$
9: $\widehat{U}_\beta \leftarrow U_{[(\tilde{n}_{\acute{s}+1}):(\tilde{n}_{\acute{s}}-1),\ (\tilde{u}_{\acute{s}+1}):(\tilde{u}_{\acute{s}}-1)]}$
10: $\widehat{R}_\alpha \leftarrow \textsc{IncCholesky}(\widehat{U}_\alpha^*\widehat{U}_\alpha)$
11: $\widehat{W}_\alpha \leftarrow W_{[0:(\tilde{n}_{s+1}-1),\ 0:(\tilde{n}_{s+1}-1)]}$
12: $\widehat{W}_\beta \leftarrow W_{[(\tilde{n}_{\acute{s}+1}):(\tilde{n}_{\acute{s}}-1),\ (\tilde{n}_{\acute{s}+1}):(\tilde{n}_{\acute{s}}-1)]}$
13: $\widetilde{W} \leftarrow \widehat{U}_\alpha(\widehat{R}_\alpha \setminus \widehat{R}_\alpha^* \setminus (-\widehat{U}_\beta^*\widehat{W}_\beta))$
14: $\widetilde{W} \leftarrow \widehat{U}_\alpha(\widehat{R}_\alpha \setminus \widehat{R}_\alpha^* \setminus (-\widehat{U}_\beta^*\widetilde{W}^*))$
15: $W \leftarrow \widehat{W}_\alpha + \widetilde{W}^*$ ▷ $\tilde{n}_{\acute{s}+1} \times \tilde{n}_{\acute{s}+1}$ *matrix*
16: **end for**
17: $D \leftarrow \mathrm{diag}(\mathrm{abs}(W)1)$
18: **return** (D, W)

3.1. Hardware Parallelism

Ignoring our multigrid strategy, using randomized matrix approximation techniques for eigenproblems has the same computational complexity as traditional eigensolvers, but with a slightly larger constant factor. However, these techniques are better suited to parallel implementations. We inherit this property; each step of our eigensolver operates simultaneously across at least m vectors.

Efficiently parallelizing a Lanczos eigensolver for running the gPb algorithm [1] on a GPU requires assuming the affinity matrix has a repeating stencil structure [3]. Our eigensolver is parallelizable without any restrictions on the sparsity patterns in the problem definition. This is important as recent work relies on solving Angular Embedding problems with data-dependent sparsity patterns [9, 10].

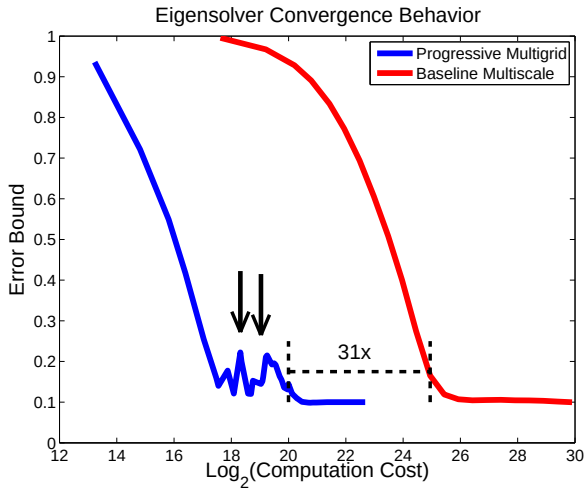

Figure 4. **Progressive multigrid speeds convergence.** For any fixed computational cost, running our solver using a progressive multigrid strategy produces significantly lower error than running it on the multiscale problem without using multigrid. Jumps in error (arrows) occur as the multigrid solver switches levels. Here, cost is the total amount of computation required for diffusion and projection operations, adjusted for the fact that operations are cheaper on coarser subproblems when using progressive multigrid. Note the log scale; multigrid is 31 times faster by this measure.

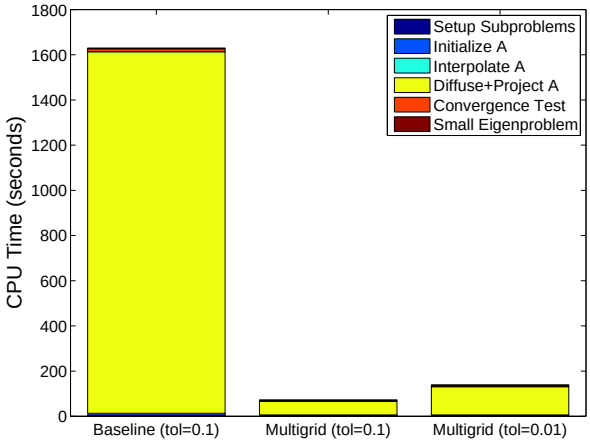

Figure 5. **Eigensolver runtime breakdown.** Diffusion and projection operations applied to refine the matrix approximation dominate the running time of our solver in both baseline and progressive multigrid modes. Even with a tighter error tolerance, the multigrid strategy offers an $11\times$ speedup over the baseline. Computation is for 32 multiscale eigenvectors. Times are averaged over the 100 BSDS [12] test images.

Figure 6. **Multiscale spectral Pb.** Following the procedure of Arbeláez *et al*. [1], we take gradients of eigenvectors to turn the result of spectral clustering into a spectral probability of boundary (sPb) measure. Our eigenvectors live on an image pyramid, rather than a grid, so we end up with a set of consistent coarse-to-fine boundaries across three scales. Comparing our results (center columns) to the original sPb (right) shows the advantage of preserving multiscale information throughout the spectral clustering stage. *Row 1:* The original sPb places an incorrect edge inside the arch and fails to pop out the top-left corner from the background. *Row 2:* Our multiscale version correctly separates the right side of the bird's head from the background; the corresponding original sPb edge is extremely weak. *Rows 3, 4:* Tiger and zebra stripes behave as they should, emerging at fine scale and disappearing at coarse. The original sPb must trade off representing object boundaries vs interior details, compressing both into a single output. *Rows 5-7:* Foreground objects pop-out strongly at coarse scale; their boundaries are more salient across scales. *Row 8:* Multiscale preserves salient structure in the left side of the image.

4. Experiments

We apply our eigensolver to image segmentation problems, defined on a multilevel pyramid [5], with scale-dependent affinities on each level. Intervening contour, computed on top of probability of boundary [1], determines affinities at three different scales. Unlike Arbeláez *et al*. [1], we refrain from collapsing the affinity matrix into a single scale before spectral clustering. We preserve multiscale information through the entire segmentation pipeline.

Figure 3 provides a visual comparison of eigensolver convergence behavior on an example image segmentation problem. Figures 4 and 5 quantify the speedup in terms of both counting operations and recording actual CPU runtime. Note that the solver spends the vast majority of time on applying the inherently m-way parallel diffusion and projection operations to matrix A, as shown by the yellow blocks in Figure 5.

Though our eigensolver is amenable to parallelization, our experiments are all on a serial implementation in MATLAB and observed speedups are solely due to our use of constraints to shape multigrid computation. Progressive multigrid gives more than a factor of 10 speedup over the baseline. All benchmarks are for finding 32 eigenvectors and are averaged over the 100 images of the Berkeley segmentation dataset test set [12].

Figure 6 compares the output from our multiscale spectral clustering problem to the single scale currently used in gPb. Clear differences hint at future applications of our multiscale pipeline to improving image segmentation quality.

5. Conclusion

Our novel eigensolver merges constrained spectral clustering with progressive multigrid computation. We demonstrate large speedups in solving multiscale image segmentation problems. Our eigensolver is applicable to many problems with coarse-to-fine structure and our segmentation framework shows the benefits of a full multiscale pipeline.

Acknowledgments. ONR MURI N00014-10-1-0933 and ARO/JPL-NASA Stennis NAS7.03001 supported Michael Maire's work. NSF CAREER IIS-1257700 supported Stella Yu's work.

References

[1] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 2011.

[2] M. Brezina, T. Manteuffel, S. McCormick, J. Ruge, G. Sanders, and P. Vassilevski. A generalized eigensolver based on smoothed aggregation (GES-SA) for initializing smoothed aggregation multigrid (SA). *Numerical Linear Algebra with Applications*, 15(2-3):249–269, 2008.

[3] B. Catanzaro, B.-Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. Efficient, high-quality image contour detection. *ICCV*, 2009.

[4] C. Chennubhotla and A. D. Jepson. Hierarchical eigensolver for transition matrices in spectral methods. *NIPS*, 2005.

[5] T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. *CVPR*, 2005.

[6] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *PAMI*, 2004.

[7] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIREV*, 2011.

[8] D. Kushnir, M. Galun, and A. Brandt. Efficient multilevel eigensolvers with applications to data analysis tasks. *PAMI*, 2010.

[9] M. Maire. Simultaneous segmentation and figure/ground organization using angular embedding. *ECCV*, 2010.

[10] M. Maire, S. X. Yu, and P. Perona. Object detection and segmentation from joint embedding of parts and pixels. *ICCV*, 2011.

[11] S. Maji, N. K. Vishnoi, and J. Malik. Biased normalized cuts. *CVPR*, 2011.

[12] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *ICCV*, 2001.

[13] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442:810–813, 2006.

[14] J. Shi and J. Malik. Normalized cuts and image segmentation. *PAMI*, 2000.

[15] S. X. Yu. Segmentation induced by scale invariance. *CVPR*, 2005.

[16] S. X. Yu. Angular embedding: A robust quadratic criterion. *PAMI*, 2012.

[17] S. X. Yu and J. Shi. Segmentation given partial grouping constraints. *PAMI*, 2004.