

# Modern Preconditioned Eigensolvers for Spectral Image Segmentation and Graph Bisection

**Andrew V. Knyazev**

*Department of Mathematics*

*Center for Computational Biology*

*Center for Computational Mathematics*

*University of Colorado at Denver*

UC-Davis Workshop, Feb 21 2006

*Supported by the NSF and the Intelligence Technology Innovation Center through the joint "Approaches to Combat Terrorism" Program Solicitation NSF 03-569.*

Center for Computational Mathematics, University of Colorado at Denver

Original image

$$B^{-1}(A - \text{stick figure})u = 0$$

Spectral segmentation 153.1405 sec

$$B^{-1}(A - \text{stick figure})u = 0$$

## Abstract

Known spectral methods for graph bipartition and image segmentation require numerical solution of eigenvalue problems with the graph Laplacian. We discuss several modern preconditioned eigenvalue solvers for computing the Fiedler vectors of large scale eigenvalue problems. The ultimate goal is to find a method with a linear complexity, i.e. a method with computational costs that scale linearly with the problem size. A locally optimal block preconditioned conjugate gradient method (LOBPCG), suggested by the speaker earlier, may be a promising candidate (if matched with a high quality preconditioner), compared against the Lanczos method. We provide preliminary numerical results, e.g., we show that a Fiedler vector of a 24 megapixel image can be computed in seconds on BlueGene/L 1024 CPU box using our BLOPEX software with Hypre preconditioning.

## OUTLINE:

1. Weighted Undirected Graphs
2. Image Segmentation Reduced to Graph Partitioning
3. Spectral Graph Partitioning
4. Traditional eigenvalue solvers for large scale problems
5. Preconditioned eigenvalue solvers
6. What is LOBPCG?
7. Block iterations in LOBPCG
8. Comparison of Features with Lanczos and Block Lanczos
9. What is LOBPCG?
10. Preliminary numerical results and Availability of LOBPCG Software

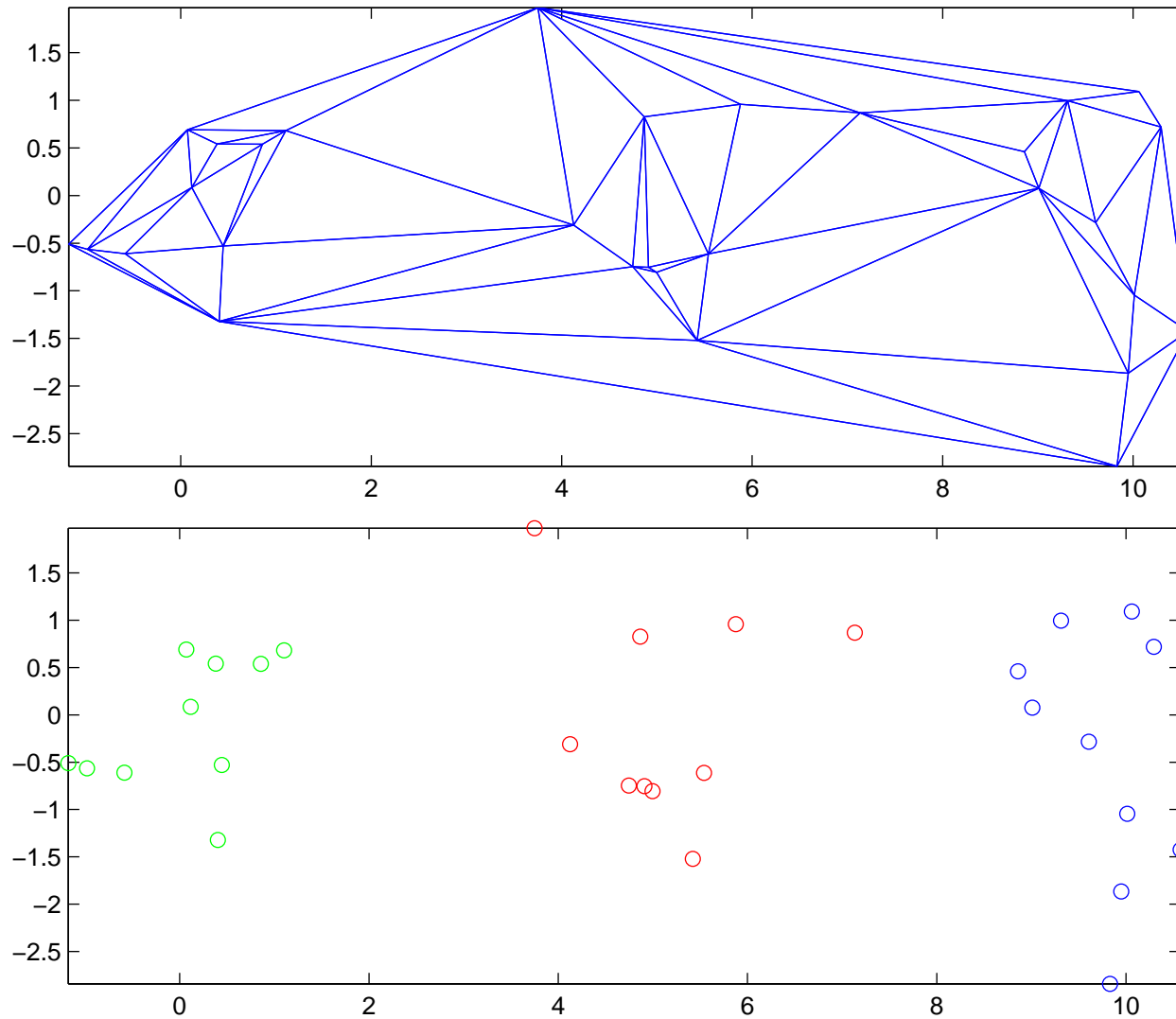
## Weighted Undirected Graphs

The set of points in an arbitrary feature space is presented as a weighted undirected graph  $G = (V, E)$ . Nodes of the graph are the points in the feature space. An edge is formed between every pair of nodes and the weight on each edge  $W(i, j)$  is a function of the similarity between nodes  $i$  and  $j$ . A graph  $G = (V, E)$  is partitioned into two disjoint complementary sets  $A$  and  $B$ ,  $B = V - A$ , by removing the edges connecting two parts.



# Spectral Image Segmentation

Andrew Knyazev, CU-Denver



## Image Segmentation Reduced to Graph Partitioning

For a given image, we construct the graph  $G = (V, E)$  by taking each pixel as a node and defining the edge weight using a special function of similarity of different pixels. This weighted graph serves as a global image feature descriptor. Image segmentation is thus reduced to graph partitioning.

It is too expensive to compare each pixel against each other pixel on the picture. Usually, we only compare neighboring pixels.

## Spectral Graph Partitioning

Several computationally efficient approaches to clustering large sparse and high-dimensional data are based on finding a few extreme eigenvalues and corresponding eigenvectors of large sparse symmetric matrices. A problem of graph bisection can be mathematically formulated as a problem of minimization of normalized cuts (Ncuts). One practical approach for approximate solution of the Ncuts problem involves numerical solution of the following eigenvalue problem

$$(D - W)y = \lambda Dy, \quad (1)$$

where  $W$  is the graph association matrix with the zero diagonal and  $D$  is a diagonal matrix defined as a row sum of  $W$ . The eigenvector, corresponding to the second smallest eigenvalue, is called the Fiedler vector. The signs of its components determine the graph bipartition.

The size of the eigenvalue problem equals to the total number of pixels of the given picture. For modern mega-pixel digital cameras, the size, therefore, is enormous. If we only compare neighboring pixels, the matrices are sparse.

Numerical solution of large scale eigenvalue problems poses major computational challenges. An efficient choice of a method for numerical solution of the eigenvalue problem becomes crucial. The ultimate goal could be to find a method with a linear complexity, in other words, a method with computational costs that scale linearly with the problem size.

## Traditional eigenvalue solvers for large scale problems

The classical Lanczos method is an evident candidate for an eigenvalue solver of choice to compute the second smallest eigenpair of (1). Several software implementations of the Lanczos method are publicly available and are well maintained, e.g., ARPACK. The traditional Lanczos method requires the generalized eigenproblem (1) be reduced, at least implicitly, to a regular eigenvalue problem for a symmetric matrix. Since the matrix  $D$  is diagonal, an obvious inexpensive possibility is to apply the Lanczos method to matrix

$$A = (\sqrt{D})^{-1}(D - W)(\sqrt{D})^{-1}. \quad (2)$$

The only, but significant, difficulty in this approach is that the smallest eigenvalues may not well relatively separated; therefore, the convergence of the Lanczos method may be slow. Even worse, by analogy with the case of the finite difference approximation of the Laplacian operator, we might expect that the separation gets worse with the increase in the problem size; thus, we do not likely achieve the linear complexity in this way.

For a natural image with  $10^5$  pixels we already observe the relative separation  $10^{-6}$  and less, that is computing just one accurate digit by the traditional Lanczos method might take thousand iterations, which is absolutely impractical.

Another well-known possibility is to apply the Lanczos method to the so-called shift-and-invert matrix, e.g., in case of eigenproblem (1) to the matrix

$$A = \sqrt{D}(D(1 - \alpha) - W)^{-1}\sqrt{D}, \quad (3)$$

where  $\alpha$  is a properly chosen small positive shift.

These matrices are not, of course, formed explicitly. Instead, each time the Lanczos method requires a multiplication of a vector  $f$  by matrix (3) a linear solver subroutine is called to solve the corresponding linear systems. If these linear systems are solved sufficiently accurately, the convergence of the Lanczos method is typically much faster compared to that when the matrix (2) is used in the Lanczos method. The difficulty now is that accurate numerical solution of linear systems, needed on each iteration of the Lanczos method, can be costly.

## Preconditioned eigenvalue solvers

In the present talk, we discuss an alternative class of methods, namely, the preconditioned eigensolvers. With an appropriate selection of a preconditioning technique, such methods can converge as fast as the Lanczos method, applied to the matrix (3), at the computational costs per iteration comparable with that of the Lanczos method, applied to the matrix (2), thus, allowing to find a path between Scylla of the slow convergence and Charybdis of expensive linear solves.

We concentrate in this talk on one of the most promising preconditioned eigensolvers, namely, on the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method, which has earlier been suggested and analyzed by the speaker. In LOBPCG for computing a single eigenpair of matrix  $A$ , the new iterate is determined by the Rayleigh–Ritz method on a three-dimensional subspace, which includes the previous iterate in addition to the current iterate and the preconditioned residual of the two-dimensional trial subspace of the steepest descent method. To compute the second eigenpair of  $A$ , the iterations are performed in the orthogonal complement to the known null space. If we need more than one eigenpair, we can compute them simultaneously.

## What is LOBPCG?

The method combines *robustness and simplicity* of the steepest descent method with a *three-term recurrence* formula:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} x^{(i-1)},$$

$$w^{(i)} = Ax^{(i)} - \lambda^{(i)} x^{(i)}, \quad \lambda^{(i)} = \lambda(x^{(i)}) = (x^{(i)}, Ax^{(i)}) / (x^{(i)}, x^{(i)})$$

with properly chosen scalar iteration parameters  $\tau^{(i)}$  and  $\gamma^{(i)}$ . The easiest and most efficient choice of parameters is based on an idea of *local optimality* Knyazev 1986, namely, select  $\tau^{(i)}$  and  $\gamma^{(i)}$  that minimize the Rayleigh quotient  $\lambda(x^{(i+1)})$  by using the *Rayleigh–Ritz method*.

$$\boxed{\text{Three-term recurrence}} + \boxed{\text{Rayleigh–Ritz method}} = \boxed{\text{Locally Optimal Conjugate Gradient Method}}$$

## Block iterations in LOBPCG

A well known idea of using *Simultaneous, or Block Iterations* provides an important improvement over single-vector methods, and permits us to compute an  $m > 1$  dimensional invariant subspace, rather than one eigenvector at a time. It can also serve as an acceleration technique over a single-vector methods on parallel computers, as convergence for extreme eigenvalues usually increases with the size of the block, and every step can be naturally implemented on wide varieties of *multiprocessor computers* as well as to take advantage of high level *BLAS* libraries.

The LOBPCG Method, is a straightforward generalization of the single-vector version enhanced with the Rayleigh–Ritz procedure on a larger trial subspace.

## Comparison of Features with Lanczos and Block Lanczos

Property	LOBPCG	Lanczos	B. Lanczos
Optimal convergence	Almost	Yes	Yes
Superlinear convergence	No	Yes	Yes
Missing eigenpairs	No	May be	No
Using block vectors BLAS	Yes	No	Yes
Algorithm simplicity	Yes	No	No
Stability/Extra costs	Low	High	Higher
Extra costs for eigenvectors	None	High	Partially
Recursive use	Yes	No	Partially
Inexact inverse/preconditioning	Yes	No	No

## What is LOBPCG?

The previous version with no preconditioning:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)}x^{(i)} + \gamma^{(i)}x^{(i-1)},$$

$$w^{(i)} = Ax^{(i)} - \lambda^{(i)}x^{(i)}, \quad \lambda^{(i)} = \lambda(x^{(i)}) = (x^{(i)}, Ax^{(i)}) / (x^{(i)}, x^{(i)})$$

The new version with the  $B^{-1}$  preconditioner:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)}x^{(i)} + \gamma^{(i)}x^{(i-1)},$$

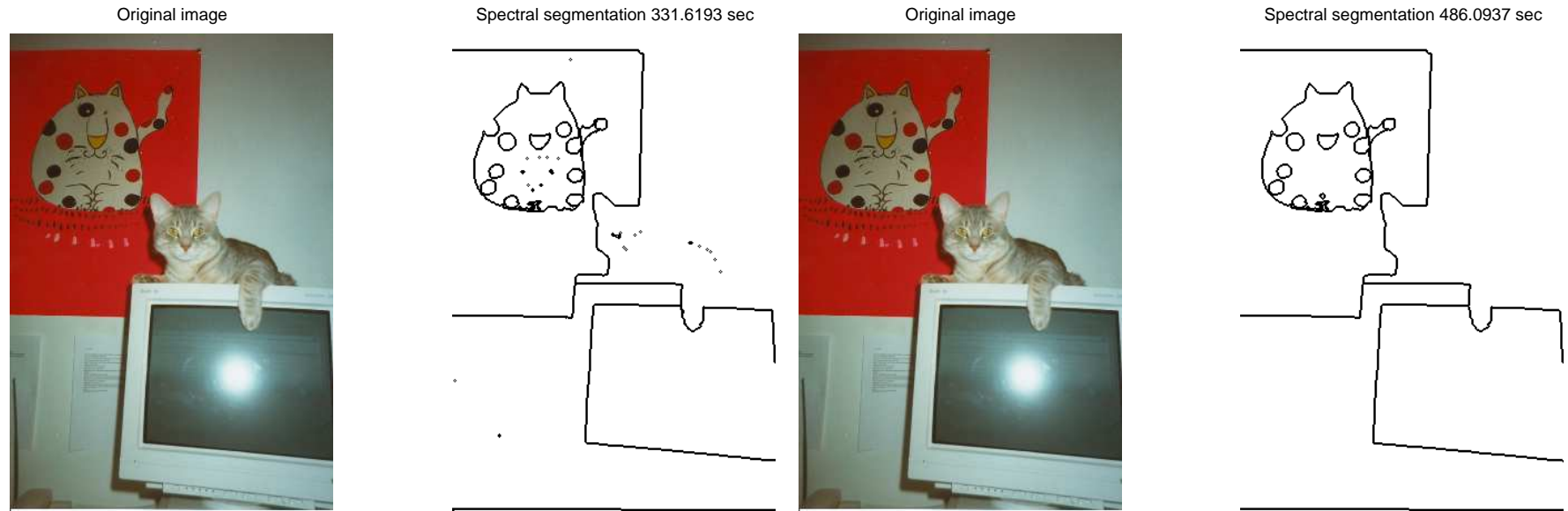
$$w^{(i)} = B^{-1}(Ax^{(i)} - \lambda^{(i)}x^{(i)}), \quad \lambda^{(i)} = \lambda(x^{(i)}) = (x^{(i)}, Ax^{(i)}) / (x^{(i)}, x^{(i)})$$

## Preliminary numerical results

We report here only limited and preliminary numerical results, since this project is a work in progress.

A recently publicly released MATLAB Graph Analysis Toolbox by Leo Grady available for download at <http://www.mathworks.com> provides the image-matrix interface for the spectral partitioning. We first test MATLAB's built-in eigensolver EIGS, based on ARPACK, against the MATLAB code of LOBPCG, using the shift-and-invert strategy.

Direct solvers in the shift-and-invert approach turn out to be practically efficient for 2D images that are not too big. The main limitation is the memory for the factors.



Segmentations of a natural  $291 \times 433$  image using LOBPCG and EIGS, both applied to matrix (3) with  $\alpha = 10^{-7}$  with no preconditioning. The MATLAB's built-in direct sparse linear solver (UMFPack) is employed and the number of iterations is capped by three by both codes. Memory use topped slightly above 2GB. LOBPCG is faster with no quality loss.

In our latest tests, we investigate Hypre algebraic multigrid preconditioning and run tests on massively parallel computers, using our LOBPCG C code, which is a part of the software library Block Locally Optimal Eigenvalue Solvers (BLOPEX). On BlueGene/L 1024 CPU we can compute the Fiedler vector of a 24 megapixel image in seconds (including the algebraic multigrid setup).

The Hypre default algebraic multigrid does not work well for the N-cut, but is extremely efficient for the standard eigenproblem. The diagonal scaling in the N-cut involves highly varying values. We may need to adjust the formula that computes the correlation between the pixels.

## Software for the preconditioned eigensolvers

The LOBPCG is publicly available in MATLAB, see <http://math.cudenver.edu/~aknyazev/software/CG/> and in C using MPI libraries for massively parallel computers. It is built-in in Hypre, see <http://www.llnl.gov/CASC/hypre/>, and is included as an external package in PETSc, see <http://www-unix.mcs.anl.gov/petsc/>.

## Conclusion

LOBPCG may be a valuable alternative to the classical Lanczos method for spectral graph partitioning and image segmentation.

Questions that need better answers:

1. How to make the algebraic multigrid to work for the N-cut?
2. What accuracy is needed from the Fiedler vectors to produce good quality partitioning?
3. How to produce stable and fast  $k$ -partitioning out of  $k$  Fiedler vectors?