SAND2013-4789C

Exceptional service in the national interest

Sandia National Laboratories

Preconditioners for Large Scale-free Graphs

Preconditioning'13, Oxford, June 19-21 2013

Erik Boman, Kevin Deweese, Richard Lehoucq

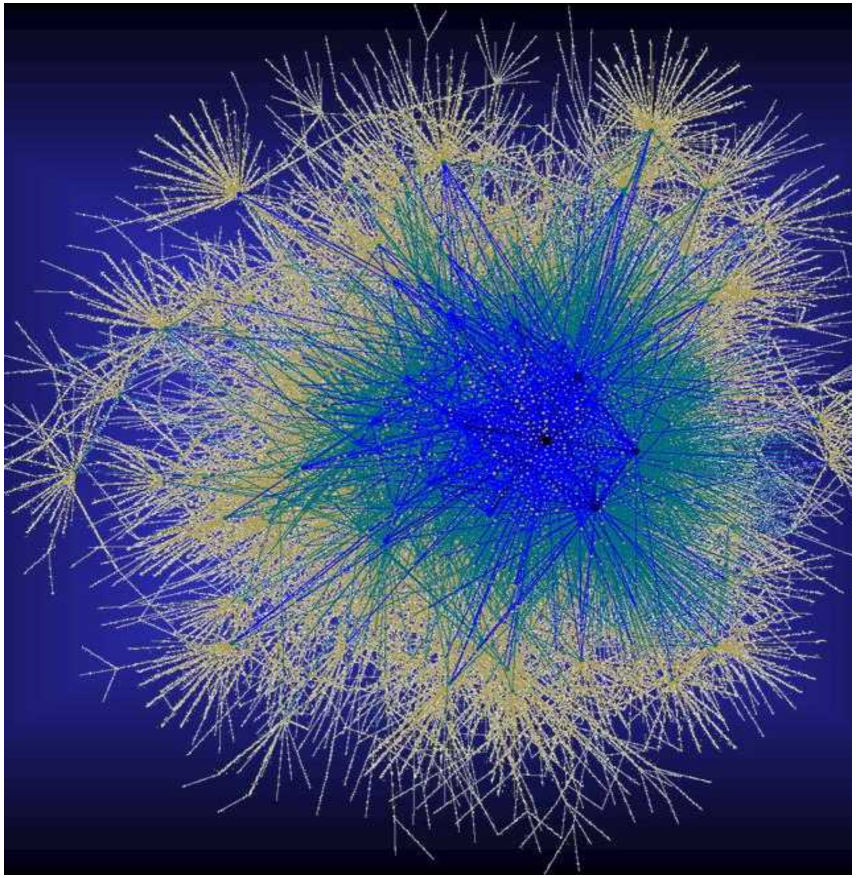
U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Background

- Large graphs are pervasive
  - WWW, social networks
- Often scale-free
  - Power-law degree distr.
  - Small diameter
- Very different from PDE discretizations
  - Need to adapt scientific computing methods and tools?



BGP graph (credit: Ross Richardson, Fan Chung)
http://math.ucsd.edu/~fan/graphs/gallery

Our Focus: Graph Laplacians

- The combinatorial Laplacian of a graph G is the sparse matrix $L(G) = D-A$, where
  - A is the adjacency matrix of G
  - $D= \mathrm{diag}(d)$ contains the degrees of the vertices in G
- We wish to:
  - Solve linear systems: $Lx=b$
  - Compute the extreme eigenpairs: $Lx = \lambda x$
- Applications:
  - Network analysis, community detection
  - Also used in spectral partitioning and ordering

More Laplacians

Several variations are of interest:

- Combinatorial Laplacian: $L_C = D - A$
- Normalized Laplacian: $L_N = D^{-1/2} (D-A) D^{-1/2}$
- Signless Laplacian: $L_S = D + A$

- We only consider the unweighted case (all edges equal)
  - But some applications have edge weights.
  - Laplacian definition and algorithms generalize naturally.

Preconditioners

- Graph Laplacians are SPSD. Typically singular but with small, known null-space.
- Little work specifically for scale-free graphs.
- Baseline is “black-box” algebraic preconditioners:
  - Jacobi (diagonal)
  - Symmetric Gauss-Seidel (SGS)
  - Incomplete Cholesky (IC)
- Traditional multigrid does not work well
  - Recent progress: LAMG (Livne & Brandt, 2012)

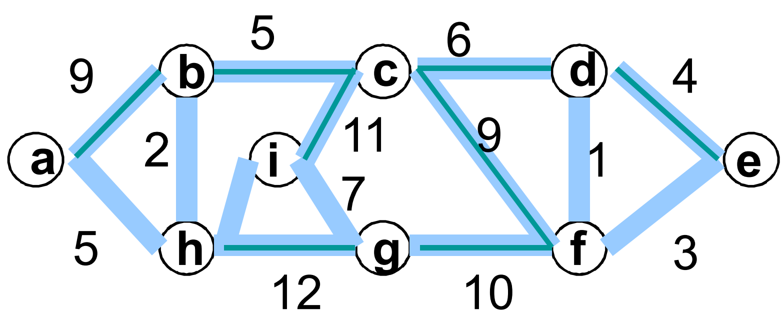
Support-Graph Preconditioners

Core Idea: Construct a sparser subgraph that is a good spectral approximation, use this as preconditioner.

- Typically, use spanning tree + “a bit more”
- First proposed by Vaidya (‘90) but not published
- Described and analyzed in [Bern et al. ‘06] and implemented by [Chen and Toledo, ‘03]
- Support theory extensions [B., Hendrickson, ‘03]
- Much recent work in theoretical CS community
  - Near optimal solvers by Spielman et al. and also by Koutis, Miller et al.
  - However: mostly theory, very few experiments
  - No software (except one Matlab code by Koutis)

Max-weight Spanning Tree (MST)

- Example:



- Efficient algorithms (Kruskal, Prim) run in ≈O(n log m) time
- Factor with no fill
- Other trees may be better, but more expensive to find
  - Low-stretch spanning trees
  - For small-world graphs, any tree has low stretch

MSF(k): A New Simple Preconditioner

- Problem: MST is often not a good preconditioner, need to add more edges.
- Two main strategies:
  - A) Carefully add edges such that fill in Cholesky factor is minimal
  - B) Add most “important” edges without regard to fill
  - Both result in very complicated algorithms!
- Simple idea: MSF(k) – Union of k spanning trees (forests)
  - M=0
  - For i=1:k
  - Find T= MST(G); G=G\T; M=M+T;
  - End

  Knob: Better (more expensive) preconditioner as k increases

Test Graphs

- Real-world graphs/mat rices from public sources:
  - UF and SNAP collections
- Symmetrized (if needed), largest component

Matrix/graph	Rows	#nonzeros
USpowergrid	5 K	18 K
Enron	68 K	507 K
Dblp-2010	226 K	1433 K
Flickr	820 K	9830 K

Results: Combinatorial Laplacian

- Test graph: USpowergrid (5K rows)
- PCG in Trilinos/Belos
- Tol = 1e-8 (rel. residual)

Precond.	Iter.	Memory (rel.)	Time(setup)	Time(solve)	Time(total)
None	883	1.0	0	.15	.15
Jacobi	431	1.0	0	.11	.11
MSF(1)	218	1.2	.02	.08	.10
IC(0)	186	1.5	.01	.07	.08
MSF(2)	59	1.9	.03	.03	.06
MSF(4)	8	2.8	.04	.01	.05

Results: Combinatorial Laplacian

- Test graph: enron (68K rows)
- PCG in Trilinos/Belos
- Tol = 1e-8 (rel. residual)

Precond.	Iter.	Memory (rel.)	Time(setup)	Time(solve)	Time(total)
None	2263	1.0	0	11.3	11.3
MSF(3)	46	2.8	3.5	0.8	4.3
MSF(2)	51	1.9	1.3	0.8	2.1
Jacobi	194	1.0	0	1.9	1.9
IC(0)	171	1.5	0.0	1.0	1.0
MSF(1)	93	1.2	0.1	0.8	0.9

Results: Normalized Laplacian

- Test graph: enron (68K rows)
- PCG in Trilinos/Belos
- Tol = 1e-8 (rel. residual)

Precond.	Iter.	Memory (rel.)	Time(setup)	Time(solve)	Time(total)
none	189	1.0	0	0.4	0.4
IC(0)	166	1.5	0.1	0.5	0.6
MSF(1)	93	1.2	0.0	0.3	0.3
MSF(2)	54	1.9	1.7	0.5	2.2
MSF(3)	47	2.8	4.3	0.6	4.9

Eigensolvers

- We use the Anasazi package in Trilinos
  - Baker, Hetmaniuk, Lehoucq, Thornquist
- Compare 4 algorithms:
  - Block Davidson (BD)
    - Davidson 1975
  - Block Krylov Schur (BKS)
    - Stewart 2000
  - LOBPCG
    - Knyazev 2001
  - Implicit Riemannian Trust-Region (IRTR)
    - Baker & Gallivan 2006
- We use block size = nev (#eigenvalues)
  - Except for BKS where 1 works better
- Preconditioning not supported in Krylov-Schur

Comparison of Eigensolvers

- Test graph: enron
- Nev = blocksize = 10 (BKS: blocksize=1)
- Tol = 1e-5 (absolute)
- 64 cores

Method	Lapl.	Prec.	Matvec.	Total time
BKS	Comb.		>50000	--
BD	Comb.	Jacobi	>50000	--
IRTR	Comb.	Jacobi	18630	33.7
LOBPCG	Comb.	Jacobi	5210	18.1
BD	Norm.		122500	238.3
IRTR	Norm.		6430	7.7
LOBPCG	Norm.		2710	7.3
BKS	Norm.		694	2.8

Comparison of Eigensolvers

- Test graph: Dblp-2010
- Nev = blocksize = 10 (BKS: blocksize=1)
- Tol = 1e-5 (absolute)
- 64 cores

Method	Lapl.	Prec.	Matvec.	Total time
BD	Comb.	Jac.	>50000	--
LOBPCG	Comb.	Jac.	*	*
BKS	Comb.		11390	114.3
IRTR	Comb.	Jac.	18490	82.8
BD	Norm.		>50000	--
IRTR	Norm.		11460	36.2
LOBPCG	Norm.		4040	27.8
BKS	Norm.		1491	8.8

Preconditioning Results: Combinatorial Laplacian

- Test graph: enron_bsl (67K rows, 507K nonzeros, 1 connected component)
- Solver=LOBPCG
- Nev = blocksize = 5
- Tol = 1e-5 (absolute)

Precondition	Lapl.	Iter.	Matvec.	Setup time	Iterate time	Total time
None	Comb.	6896	34505	-	564.8	564.8
Jacobi	Comb.	375	1890	0.0	54.3	54.3
IC(0)	Comb.	217	1100	0.1	32.0	32.1
SGS	Comb.	155	780	0.0	21.4	21.4
MSF(1)	Comb.	125	630	1.9	13.2	15.1
MSF(3)	Comb.	44	245	8.2	6.5	14.7
MSF(2)	Comb.	53	270	3.3	6.2	9.5

t

Preconditioning Results: Normalized Laplacian

- Test graph: enron_bsl (67K rows, 507K nonzeros, 1 connected component)
- Solver=LOBPCG
- Nev = blocksize = 5
- Tol = 1e-5 (absolute)

Precond.	Laplacian	Iter.	Matvec.	Setup time	Iterate time	Total time
IC(0)	Norm.	270	1020	0.1	37.4	37.5
None	Norm.	194	1005	0	25.8	25.8
MSF(3)	Norm.	34	195	18.9	6.5	25.4
MSF(2)	Norm.	42	215	7.9	6.2	14.1
MSF(1)	Norm.	102	515	2.0	10.7	12.7
SGS	Norm.	87	440	0.0	11.8	11.8

Conclusions

- Computations on scale-free graphs are different than PDE discretizations
  - Graph-based preconditioners make sense on such graphs
- MSF(k) is a simple but effective preconditioner
  - K=2 may work better than k=1, at least on small problems
- Normalized Laplacians are computationally easier
- Preconditioning is essential for combinatorial Laplacian
  - Also helps a bit for normalized Laplacian
- Eigensolver: BKS best for normalized Laplacian
  - No preconditioning needed for normalized problems?
  - No clear winner among {BKS, LOBPCG, IRTR} for combinatorial
- Work in progress: Larger problems on parallel computers
  - Need to revisit preconditioners (domain decomposition)

Extra Slides

Results: 2D Grid

- 100 x 100 regular grid (toy problem)
- PCG
- Tol = 1e-6 (residual)

Precond.	Iter.	Memory (rel.)	Flops(setup)	Flops(solve)	Flops(total)
Jacobi	345	1.0	0	19.9M	19.9M
IC(0)	146	1.6	0.1M	16.2M	16.3M
MSF(1)	186	1.4	0.05M	16.6M	16.6M
MSF(2)	13	4.5	23.2M	5.2M	28.4M
MSF(3)	4	5.1	29.5M	1.4M	30.8M
Cholesky	1	5.1	29.9M	0.4M	30.3M