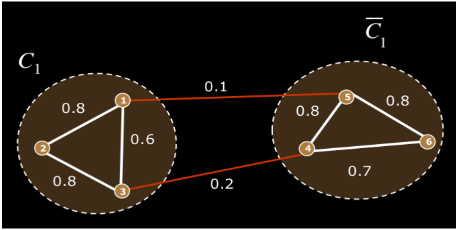
Practical Computation of the Fiedler Vector in a Single Processor

Darío Bahena

CINVESTAV

June, 2017

The Problem: Minimal Bi-Partitional RatioCut



- Can be seen as an NP-Hard discrete optimization problem (where $G = (V, W)$ is the graph represented by W):

$$\min_{C_1 \subset V} \underbrace{\frac{1}{2}\left[\frac{\mathrm{cut}(C_1, \overline{C_1})}{|C_1|} + \frac{\mathrm{cut}(\overline{C_1}, C_1)}{|\overline{C_1}|}\right]}_{\mathrm{RatioCut}(C_1, \overline{C_1})} \ni \quad \mathrm{cut}(A, B) = \sum_{i \in A\ ,\ j \in B} w_{ij}$$

The Laplacian and its Fiedler Vector

Approximate by removing discrete constraints (signs)

$$\min_{C_1 \subset V} \text{RatioCut}(C1, \overline{C1}) \quad \equiv \quad \min_{\mathbf{f} \in \mathbb{R}^n} \underbrace{\left(\frac{\mathbf{f}^T \, L \, \mathbf{f}}{\mathbf{f}^T \, \mathbf{f}} \right)}_{\text{Rayleigh-Quotient}} \quad \ni \quad L = D - W \, \wedge \, \mathbf{f} \perp \mathbf{1}$$

Courant-Fischer Theorem (for λ_2 and A real-symmetric)

$$\lambda_2(A) = \max_{\dim(U) = n-1} \left[\min_{\mathbf{x} \in U \, \wedge \, \|\mathbf{x}\| \neq 0} \left(\frac{\mathbf{x}^T \, A \, \mathbf{x}}{\mathbf{x}^T \, \mathbf{x}} \right) \right]$$

Problem reduces to finding Fiedler Vector of L

$$\min_{\mathbf{x} \perp \mathbf{1} \, \wedge \, \|\mathbf{x}\| \neq 0} \left(\frac{\mathbf{x}^T \, L \, \mathbf{x}}{\mathbf{x}^T \, \mathbf{x}} \right) = \lambda_2 \; \ni \; L\mathbf{x} = \lambda_2 \mathbf{x} \quad (L \text{ is SSPSD, leverage that!})$$

This thesis in a nutshell

Goal

On the context of a real-life application doing spectral clustering; the objective of this thesis is to emit an algorithm recommendation, for efficient computation of the the Fiedler Vector on a single processor (application requirement).

Algorithms Preliminaries

(Symmetric) Power Method

$$\mathbf{v}_0 = \mathbf{v} \;\wedge\; \mathbf{v}_{k+1} = A\mathbf{v}_k \left(A^{-1}\mathbf{v}_k\right) \quad \ni \quad \text{convergence} \sim \frac{\lambda_{n-1}}{\lambda_n} \left(\frac{\lambda_1}{\lambda_2}\right)$$

Krylov Subspaces

$$\mathrm{K}_m(A, \mathbf{x}_0) = \mathrm{span}\left\{A^0\mathbf{x}_0 \;,\; A^1\mathbf{x}_0 \;,\; \ldots \;,\; A^{(m-1)}\mathbf{x}_0\right\}$$

Rayleigh-Ritz Method

- Compute orthonormal basis V of subspace ($\dim(V) \ll \dim(A)$).
- Solve (smaller) eigenproblem $R\mathbf{y} = \lambda\mathbf{y} \quad \ni \quad R = V^T AV$
- Compute the Ritz pairs $(\widetilde{\lambda}_i, \widetilde{\mathbf{x}}_i) = (\lambda_i, V\mathbf{y}_i)$

Algorithms Outline (serial usage)

Scope

Just cover main ideas, usage and try to leverage Laplacian (SSPSD).

For symmetric dense matrices

- Convert into tridiagonal form first.
- Symmetric Tridiagonal QL Algorithm (all or nothing).
- MRRR: Quite sophisticated, independent eigen-pairs computation.

For symmetric sparse matrices

- Compute small side of spectra, no explicit A (just $A\mathbf{x}$).
- Lanczos (L. 1950) in ARPACK/IRLM ($\sim$ 1998).
- LOBPCG (Knyazev 2001) in SciPy/NetworkX ($\sim$ 2008).

Lanczos vs LOBPCG

Feature/Issue	Lanczos(IRLM)	LOBPCG (single)
Need Restarting	Yes, due $\mathrm{K}_m(L^{-1}, \mathbf{x}_0)$	No, span$\{\mathbf{x}_i, T\mathbf{r}_i, \mathbf{x}_{i-1}\}$
Search strategy	Plain search	$\nabla(\rho(\mathbf{x_i}))$ $\ni \ \mathbf{r}_i = L\mathbf{x}_i - \rho(\mathbf{x}_i)\mathbf{x}_i$
Search Constraints	No	Yes $(Y^\perp = \mathbf{1}^\perp)$
Requested spectra	$k = 2$	$k = 1$
Matrix operation	solve $L\mathbf{x} = \mathbf{b}$	$L\mathbf{x}$
Uses Preconditioning	Only with iter solvers	Yes $\left(T = \dfrac{1}{\mathrm{diag}(L)}\right)$
Clustered eigenvalues	Inherited from PM	Bug?

Experiment Setup

Hardware

- Intel®Core™ $i5$ at 2.40GHz (Linux taskset), 8Gb of RAM ($\ll$).

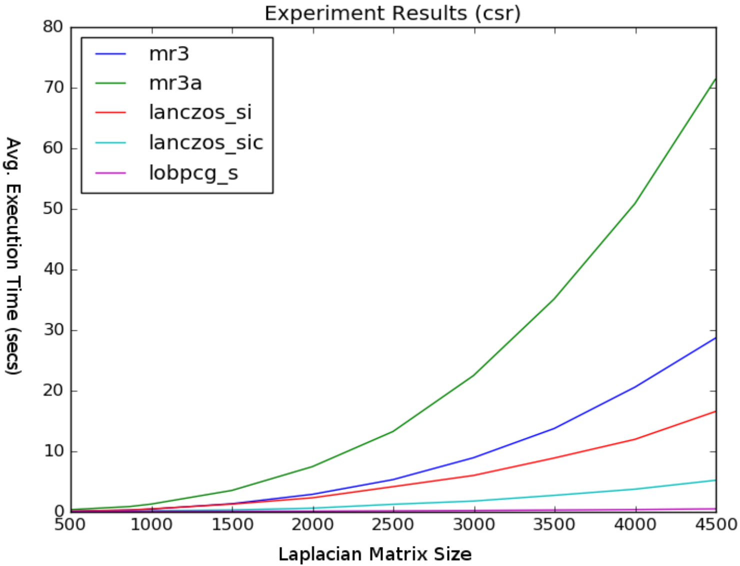
Algorithms implementations (rely on BLAS/LAPACK)

- Scipy/NetworkX Python wrappers around native libs.
- MRRR: LAPACK implementation (Fortran90).
- Lanczos: ARPACK implementation (Fortran77).
- LOBPCG: Scipy implementation (pure Python).

Data preparation

- 10 CSR/CSC matrices from app. domain (size in $[867, 4500]$).
- Removed small SCC (≤ 1.2 secs) and shifted spectra ($L + 0.01I$).
- Average time out of 100 executions (MRRR is gold standard).

Experiment Results



Conclusions

Recommendations

- Prefer sparse algorithms (even if data fits in memory).
- If forced to use dense-matrix alg., prefer MRRR (LAPACK / JNI).
- Lanczos/ARPACK is available through Java Netlib.
- If not urgent, worth it to port LOBPCG to Java (single vector).

Additional considerations

- Need to have efficient matrix representations (CSC/CSR) in Java.
- Need to port to Java SCC Algorithm (CSR).

Challenges found during thesis

- Numerical Linear Algebra is usually a graduate topic. Hard to find accessible entry-point material.
- Plenty of algorithms to explore (more time).

Thank you

Questions?

Appendix

Auxiliary slides from here

Some theory

The Laplacian

- Sparse ($\approx 70\%$).
- Symmetric.
- Positive Semi-Definite.
- Its first eigenpair is $(0, \mathbf{1})$.

The Symmetric Eigenvalue Problem

- Always has solution (Spectral Theorem: $L = Q^T LQ$).
- Ill-Conditioning (Bindel and Goodman): only on small and clustered eigenvalues.
- There are some Backward Stable Algoritm implementations in general; but from our set of three only MRRR has this formalism.

Lanczos Algorithm (ARPACK/IRLM)

Main idea

Apply Rayleigh-Ritz against $\mathrm{K}_m(A, \mathbf{x_0}) \ni m > k$. Uses dense eigen-solver for $m \times m$ symmetric matrix H.

Restarting (avoid m to grow indefinitely)

Implicitly Restarted QR Algorithm to apply p shifts against H:

$$j = 1 \ldots p : \; QR = \mathrm{qr}(H - \lambda_j I) \wedge H = \; Q^T \, HQ \quad \ni \quad m = k + p$$

Practical considerations

- We set $k = 2$ and by default $m = 2k + 1$.
- Used shift-invert mode with $\sigma = 0$, and one linear sparse solver (SuperLU or Cholmod); which accounts for $\approx$ %80 of time.
- Slow convergence for clustered eigenvalues (eg disconn. graph).

LOBPCG

Ancestor (dense eigensolver 2×2)

Preconditioned Steepest Descent Algorithm applies Rayleigh-Ritz against $\text{span}\{\mathbf{x}_i, T\mathbf{r}_i\}$ (T is a preconditioner) $\ni$

$$\mathbf{r}_i = A\mathbf{x}_i - \rho(\mathbf{x}_i)\mathbf{x}_i \;\wedge\; \rho(\mathbf{x}) = \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T \mathbf{x}}$$

Main ideas (dense eigensolver 3×3)

Uses subspace $\text{span}\{\mathbf{x}_i, T\mathbf{r}_i, \mathbf{x}_{i-1}\}$ to accelerate convergence (in practice $x_i - \beta x_{i-1}$); Knyazev also proved that T must be SPD to guarantee convergence. His algorithm also features constraints Y (search in $Y^\perp$).

Practical considerations (NetworkX)

- We set $k = 1$ and $Y = \mathbf{1}$.
- $T = \dfrac{1}{\text{diag}(A)}$ (SPD)
- Numerical errors on clustered eigenvalues.

Recomputing W after removal of small CC

Adapted from an stackoverflow.com post (COO format)

```
def split_cc_sparse2(W, cclab):
    idx_del = np.nonzero(cclab)[0]
    keep_row = np.logical_not(np.in1d(W.row, idx_del))
    keep_col = np.logical_not(np.in1d(W.col, idx_del))
    keep = np.logical_and(keep_row, keep_col)
    W.data = W.data[keep]
    W.row = W.row[keep]
    W.col = W.col[keep]
    W.row -= np.less.outer(idx_del, W.row).sum(0)
    W.col -= np.less.outer(idx_del, W.col).sum(0)
    k = len(idx_del)
    W._shape = (W.shape[0] - k, W.shape[1] - k)
    return W
```