**Spectral Graph Theory** Lecture 16

# Preconditioning by Low-Stretch Spanning Trees

*Daniel A. Spielman* October 23, 2009

## 16.1 Overview

After reviewing the most important facts about the Conjugate Gradient algorithm, I will explain how it can be improved by preconditioning. I will then show how preconditioning relates to graphic inequalities, and show that low-stretch spanning trees provide good preconditioners. This last part is an extension from [SW09] of a suggestion of Boman and Hendrickson [BH01].

## 16.2 The Conjugate Gradient, take 2

I'll begin this lecture by stating again the important properties of the Conjugate Gradient algorithm, since they might have been missed in the fog of last lecture.

The Conjugate Gradient algorithm allows us to solve linear equations of the form

$$A\boldsymbol{x} = \boldsymbol{b},$$

where $A$ is symmetric and positive definite. We can also use this algorithm if $A$ is degenerate, but we know the nullspace.

The Conjugate Gradient is iterative. In each iteration it multiplies one vector by $A$, and performs a constant number of vector operations. Let $\boldsymbol{x}^t$ be the $t$th iterate produced by the algorithm. It satisfies the following guarantee:

$$\frac{\left\|\boldsymbol{x} - \boldsymbol{x}^t\right\|_A}{\|\boldsymbol{x}\|_A} \leq \inf_{\substack{p \text{ of degree } t \\ p(0)=1}} \max_{\lambda_i \in \text{eigs}(A)} |p(\lambda_i)| .$$

I should recall that

$$\|\boldsymbol{x}\|_A = \sqrt{\boldsymbol{x}^T A \boldsymbol{x}}.$$

To get some idea of where this comes from, recall that I said that the Conjugate Gradient finds the

$$\boldsymbol{x}^t \in \textbf{Span}\left(\boldsymbol{b}, A\boldsymbol{b}, A^2\boldsymbol{b}, \ldots, A^{t-1}\boldsymbol{b}\right)$$

that minimizes

$$\left\|\boldsymbol{x} - \boldsymbol{x}^t\right\|_A .$$

Now, if $p(X)$ is a polynomial of degree $t$ such that $p(0) = 1$, we can write

$$p(X) = 1 - q(X)X,$$

where $q(X)$ is a polynomial of degree $t - 1$. The Conjugate Gradient may choose

$$\boldsymbol{x}^t = q(A)\boldsymbol{b}.$$

As $\boldsymbol{b} = A\boldsymbol{x}$, this gives

$$\boldsymbol{x}^t = q(A)A\boldsymbol{x},$$

and

$$\boldsymbol{x} - \boldsymbol{x}^t = \boldsymbol{x} - q(A)A\boldsymbol{x} = p(A)\boldsymbol{x}.$$

If we expand $\boldsymbol{x}$ in the eigenbasis of $A$ as

$$\boldsymbol{x} = \sum_i c_i \boldsymbol{v}_i,$$

then we get

$$\boldsymbol{x} - \boldsymbol{x}^t = \sum_i c_i p(\lambda_i)\boldsymbol{v}_i,$$

and so

$$\left\|\boldsymbol{x} - \boldsymbol{x}^t\right\|_A^2 = \sum_i c_i p(\lambda_i)^2 \lambda_i \boldsymbol{v}_i \leq \max_i |p(\lambda_i)|^2 \sum_i c_i \lambda_i \boldsymbol{v}_i = \left(\max_i |p(\lambda_i)|^2\right) \|\boldsymbol{x}\|_A^2 .$$

In particular, the $n$th iterate must be exactly $\boldsymbol{x}$. To see this, consider the polynomial

$$p(X) = \prod_i (1 - X/\lambda_i).$$

It takes the value 1 at 0, and is zero at every eigenvalue of $A$. So, the Conjugate Gradient always takes at most $n$ iterations. This is wonderful when $A$ is sparse, say having $O(n)$ non-zero entries. In this case, CG runs in time $O(n^2)$, which is as long as it would take just to write down the inverse of $A$.

The Conjugate Gradient is often faster than this.

**Theorem 16.2.1.** *For $t \geq 1$ and $0 < \alpha < \beta$, there exists a polynomial $p(X)$ of degree $t$ such that $p(0) = 1$ and*

$$|p(\lambda)| \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^t,$$

*for all $\alpha \leq \lambda \leq \beta$ and $\kappa = \beta/\alpha$.*

## 16.3 Preconditioning

The idea of preconditioning is to instead consider the problem of solving the linear system

$$B^{-1}A\boldsymbol{x} = B^{-1}\boldsymbol{x}.$$

This system of course has the same solution as the original. The matrix $B$ is called a *preconditioner*, and it should also be positive definite (or positive semidefinite with the same known nullspace as $A$).

We will apply a variant of the Conjugate Gradient, called the Preconditioned Conjugate Gradient, to solve this modified system. In each iteration, it will multiply a vector by $A$, solve a linear system in $B$, and perform a constant number of vector operations. Its output satisfies the same guarantee as the Conjugate Gradient, but with

$$\kappa(A, B) = \frac{\lambda_{max}(B^{-1}A)}{\lambda_{min}(B^{-1}A)}.$$

If we use $B = I$ the linear solve in $B$ is trivial, but the running time is the same as the Conjugate Gradient. If we use $B = A$, then $\kappa(A, B) = 1$, but we haven't saved ourselves any work. Preconditioning is useful when $\kappa(A, B)$ is much smaller than $\kappa(A)$, and it is easy to solve linear equations in $B$.

You might be concerned that $B^{-1}A$ is not symmetric. Let me put those concerns to rest. As $A$ is positive definite, it has a square root:

$$A^{1/2} = VD^{1/2}V^T, \quad \text{where } A = VDV^T.$$

So, $B^{-1}A$ has the same eigenvalues as

$$A^{1/2}(B^{-1}A)A^{-1/2} = A^{1/2}B^{-1}A^{1/2},$$

which is symmetric and positive definite. Similarly, for any $j$

$$A^{1/2}(B^{-1}A)^jA^{-1/2} = \left(A^{1/2}B^{-1}A^{1/2}\right)^j.$$

PCG finds the $\boldsymbol{x}^t$ in

$$\boldsymbol{x}^t \in \mathbf{Span}\left(\boldsymbol{b}, b^{-1}A\boldsymbol{b}, (B^{-1}A)^2\boldsymbol{b}, \ldots, (B^{-1}A)^{t-1}\boldsymbol{b}\right)$$

that minimizes

$$\left\|\boldsymbol{x} - \boldsymbol{x}^t\right\|_A.$$

Thus, we can show that

$$\frac{\left\|\boldsymbol{x} - \boldsymbol{x}^t\right\|_A}{\left\|\boldsymbol{x}\right\|_A} \leq \inf_{\substack{p \text{ of degree } t \\ p(0)=1}} \max_{\lambda_i \in \text{eigs}(B^{-1}A)} \left|p(\lambda_i)\right|.$$

To see why, write $p(X) = 1 - q(X)X$ as before. If we set $\boldsymbol{x}^t = q(X)\boldsymbol{b}$, then

$$\begin{aligned}\frac{\left\|\boldsymbol{x} - \boldsymbol{x}^t\right\|_A^2}{\|\boldsymbol{x}\|_A^2} &= \frac{(\boldsymbol{x} - \boldsymbol{x}^t)^T A(\boldsymbol{x} - \boldsymbol{x}^t)}{\boldsymbol{x}^T A\boldsymbol{x}} \\ &= \frac{(p(B^{-1}A)\boldsymbol{x})^T A(p(B^{-1}A)\boldsymbol{x})}{\boldsymbol{x}^T A\boldsymbol{x}} \\ &= \frac{(p(A^{1/2}B^{-1}A^{1/2})\boldsymbol{y})^T (p(A^{1/2}B^{-1}A^{1/2})\boldsymbol{y})}{\boldsymbol{y}^T\boldsymbol{y}},\end{aligned}$$

where we set

$$\boldsymbol{y} = A^{1/2}\boldsymbol{x}.$$

As before, we see that this last ratio is at most

$$\left\|p(A^{1/2}B^{-1}A^{1/2})\right\|^2 = \max_{\lambda_i \in \mathrm{eigs}(A^{1/2}B^{-1}A^{1/2})} |p(\lambda_i)|^2 = \max_{\lambda_i \mathrm{eigs}(B^{-1}A)} |p(\lambda_i)|^2 .$$

## 16.4 Approximating Graphs and Preconditioning

Let me first relate the eigenvalues of $B^{-1}A$ to our $\preccurlyeq$ notation.

**Lemma 16.4.1.** *Let $A$ and $B$ be positive definite matrices such that*

$$\alpha B \preccurlyeq A \preccurlyeq \beta B.$$

*Then all the eigenvalues of $B^{-1}A$ lie between $\alpha$ and $\beta$.*

*Proof.* We will just prove the upper bound. We have

$$\begin{aligned}\lambda_{max}(B^{-1}A) &= \lambda_{max}(B^{-1/2}AB^{-1/2}) \\ &= \max_{\boldsymbol{x}} \frac{\boldsymbol{x}^T B^{-1/2}AB^{-1/2}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} \\ &= \max_{\boldsymbol{y}} \frac{\boldsymbol{y}^T A\boldsymbol{y}}{\boldsymbol{y}^T B\boldsymbol{y}}, \qquad \text{settting } \boldsymbol{y} = B^{-1/2}\boldsymbol{x}, \\ &\leq \beta.\end{aligned}$$

□

Again, we can do all this with degenerate positive semi-definite matrices as long as we know their nullspaces. Throughout this lecture, whenever I write the inverse of a degenerate matrix, I will actually mean the pseudo-inverse. This is the operator that is the inverse on the range, and is zero on the nullspace. Formally, if $A = \sum_i \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T$, then the pseudo-inverse of $A$ is

$$\sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} \boldsymbol{v}_i \boldsymbol{v}_i^T.$$

Vaidya [Vai90] had the remarkable idea of preconditioning the Laplacian matrix of a graph by the Laplacian matrix of a subgraph. If $H$ is a subgraph of $G$, then

$$L_H \preccurlyeq L_G,$$

so all eigenvalues of $L_H^{-1} L_G$ are at least 1. This reduces us to upper bounding the largest eigenvalues, and finding subgraphs whose Laplacians are easy to invert.

## 16.5 Preconditioning by Trees

It is relatively easy to show that linear equations in the Laplacian matrices of trees can be solved exactly in linear time. One can either do this by finding an $LU$-factorization with a linear number of non-zeros, or by viewing the process of solving the linear equation as a dynamic program that passes up once from the leaves of the tree to a root, and then back down.

We will now show that a special type of tree, called a *low-stretch spanning tree* provides a very good preconditioner. To begin, let $T$ be a spanning tree of $G$. Write

$$L_G = \sum_{(u,v)\in E} w_{u,v} L_{u,v} = \sum_{(u,v)\in E} w_{u,v} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T.$$

We will actually consider the trace of $L_T^{-1} L_G$. As the trace is linear, we have

$$\begin{aligned}
\mathrm{Tr}\left(L_T^{-1} L_G\right) &= \sum_{(u,v)\in E} w_{u,v} \mathrm{Tr}\left(L_T^{-1} L_{u,v}\right) \\
&= \sum_{(u,v)\in E} w_{u,v} \mathrm{Tr}\left(L_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T\right) \\
&= \sum_{(u,v)\in E} w_{u,v} \mathrm{Tr}\left((\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T L_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)\right) \\
&= \sum_{(u,v)\in E} w_{u,v} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T L_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v).
\end{aligned}$$

To see why the last step is true, recall that $\mathrm{Tr}\,(AB) = \mathrm{Tr}\,(BA)$ for all matrices $A$ and $B$. To evalue this last term, we need to know the value of $(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T L_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)$. You already know something about it. Let $T(u,v)$ denote the path in $T$ from $u$ to $v$, and let $w_1, \ldots, w_k$ denote the weights of the edges on this path. We proved in Lemma 6.5.2 that

$$L_{u,v} \preccurlyeq \left(\sum_{i=1}^{k} \frac{1}{w_i}\right) L_{T(u,v)}.$$

As $L_{u,v}$ is a rank-1 matrix, this tells us that

$$(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T L_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v) \leq \sum_{i=1}^{k} \frac{1}{w_i}. \tag{16.1}$$

In fact, this is an equality. Even better, the term (16.1) is something that has been well-studied. It was defined by Alon, Karp, Peleg and West [AKPW95] to be the *stretch* of the unweighted edge $(u, v)$ with respect to the tree $T$. Moreover, the *stretch* of the edge $(u, v)$ with weight $w_{u,v}$ with respect to the tree $T$ is defined to be exactly

$$w_{u,v} \sum_{i=1}^{k} \frac{1}{w_i},$$

where again $w_1, \ldots, w_k$ are the weights on the edges of the unique path in $T$ from $u$ to $v$. A sequence of works, begining with [AKPW95], has shown that every graph $G$ has a spanning tree in which the sum of the stretches of the edges is low. The best result so far is due to [ABN08], who prove the following theorem.

**Theorem 16.5.1.** *Every weighted graph $G$ has a spanning tree subgraph $T$ such that the sum of the stretches of all edges of $G$ with respect to $T$ is at most*

$$O(m \log n (\log \log n)^2),$$

*where $m$ is the number of edges $G$.*

Thus, if we choose a low-stretch spanning tree $T$, we will ensure that

$$\mathrm{Tr}\left(L_T^{-1} L_G\right) = \sum_{(u,v)\in E} w_{u,v} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T L_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v) \leq O(m \log n (\log \log n)^2).$$

In particular, this tells us that $\lambda_{max}(L_T^{-1} L_G)$ is at most $O(m \log n (\log \log n)^2)$, and so the Preconditioned Conjugate Gradient will require at most $O(m^{1/2} \log n)$ iterations, each of which requires one multiplication by $L_G$ and one linear solve in $L_T$.

## 16.6 Improving the Bound on the Running Time

We can show that the Preconditioned Conjugate Gradient will actually run in closer to $O(m^{1/3})$ iterations. Since the trace is the sum of the eigenvalues, we know that for every $\beta > 0$, $L_T^{-1} L_G$ has at most

$$\mathrm{Tr}\left(L_T^{-1} L_G\right) / \beta$$

eigenvalues that are larger than $\beta$.

To exploit this fact, we use the following lemma.

**Lemma 16.6.1.** *Let $\lambda_1, \ldots, \lambda_n$ be positive numbers such that all of them are at least $\alpha$ and at most $k$ of them are more than $\beta$. Then, for every $t \geq k$, there exists a polynomial $p(X)$ of degree $t$ such that $p(0) = 1$ and*

$$|p(\lambda_i)| \leq 2 \left( \frac{\sqrt{\beta/\alpha} - 1}{\sqrt{\beta/\alpha} + 1} \right)^{t-k},$$

*for all $\lambda_i$.*

*Proof.* Let $r(X)$ be the polynomial of degree $t-k$ known to exist by Theorem 16.2.1 for which

$$|r(\lambda)| \leq 2\left(\frac{\sqrt{\beta/\alpha}-1}{\sqrt{\beta/\alpha}+1}\right)^{t-k},$$

for all $\lambda$ between $\alpha$ and $\beta$. Now, set

$$p(X) = r(X)\prod_{i:\lambda_i>\beta}(1-X/\lambda_i).$$

This new polynomial is zero at every $\lambda_i$ greater than $\beta$, and for $\lambda$ less than $\beta$

$$|p(\lambda)| = |r(\lambda)|\prod_{i:\lambda_i>\beta}|(1-\lambda/\lambda_i)| \leq |r(\lambda)|,$$

as we always have $\lambda < \lambda_i$ in the product. □

Applying this lemma to the analysis of the Preconditioned Conjugate Gradient, with $\beta = \mathrm{Tr}\left(L_T^{-1}L_G\right)^{2/3}$ and $k = \mathrm{Tr}\left(L_T^{-1}L_G\right)^{1/3}$, we find that the algorithm produces $\epsilon$-approximate solutions within

$$O(\mathrm{Tr}\left(L_T^{-1}L_G\right)^{1/3}\ln(1/\epsilon)) = O(m^{1/3}\log n \ln 1/\epsilon)$$

iterations.

## 16.7 Further Improvements

Next week, Nikhil Srivastava will prove that for every graph $G$, there exists a graph $H$ with $O(n/\epsilon^2)$ edges that is an $\epsilon$-approximation of $G$. This construction may be viewed as a far-reaching generalization of expander graphs.

As we can solve linear equations in $H$ in time $O(n^2)$, this gives us a way of solving linear equations in $G$ in time $O(n^2 \ln 1/\epsilon)$, regardless of how many edges $G$ has. This is better than $O(m^{4/3})$ when $m$ is large.

In fact, by combining low-stretch spanning trees and sparse high-quality graph approximations (called sparsifiers), one can get algorithms that solve linear systems in Laplacians in time $O(m\log^{O(1)} n)$ [ST08].

## References

[ABN08] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 781–790, Oct. 2008.

[AKPW95] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the $k$-server problem. *SIAM Journal on Computing*, 24(1):78–100, February 1995.

[BH01] Erik Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[ST08] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2008. Available at `http://www.arxiv.org/abs/cs.NA/0607105`.

[SW09] Daniel A. Spielman and Jaeoh Woo. A note on preconditioning by low-stretch spanning trees. *CoRR*, abs/0903.2816, 2009. Available at `http://arxiv.org/abs/0903.2816`.

[Vai90] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990.