Lecture 15

Graph sparsification

15.1 Solving Laplacian systems of equations

We now focus on linear systems of equations $Mx = b$ for which the matrix M has a little more structure. We'll see how these can be solved with an exponentially better dependence on the condition number than the algorithms from the previous lecture. The method relies on a short series expansion, together with a technique of sparsification used to prevent the matrices in the expansion from becoming dense.

Definition 15.1. An M-matrix is a matrix M such that each off-diagonal entry of M is nonpositive, and the row sums of M are nonnegative and not all zero.

For example, a normalized Laplacian matrix to which we added a positive number anywhere on the diagonal (to ensure not all row sums are zero) is an M-matrix. Now let's get a key formula for the inverse of an M-matrix.

Lemma 15.2. *Suppose $M = D - A$ is an M-matrix. Then*

$$(D-A)^{-1} = \frac{1}{2}\big(D^{-1} + (\mathbb{I} + D^{-1}A)(D - AD^{-1}A)^{-1}(\mathbb{I} + AD^{-1})\big). \tag{15.1}$$

Moreover, $M' = D - AD^{-1}A$ is also an M-matrix.

Note how (15.1) lets us compute the inverse of $D-A$ inductively, by computing the inverse of $(D - AD^{-1}A)^{-1}$ and performing a couple simple multiplications (the inverses D^{-1} are easy to compute since D is diagonal). The key point, as we'll see, is that the approximation quality improves in the process — if we have a good approximation to $D - AD^{-1}A$, in some precise sense to be defined later, then by plugging it into (15.1) we will get an even better approximation to $(D - A)^{-1}$. First let's prove the lemma.

Proof of Lemma 15.2. To check (15.1) is correct, multiply both sides by $(D - A)$ on the left.

On the left we get $\mathbb{I}$. On the right,

$$\begin{aligned}\frac{1}{2}(D-A)\big(D^{-1}+(\mathbb{I}+D^{-1}A)(D-AD^{-1}A)^{-1}(\mathbb{I}+AD^{-1})\big)&\\ &= \frac{1}{2}\big((\mathbb{I}-AD^{-1})+(D-A+A-AD^{-1}A)(D-AD^{-1}A)(\mathbb{I}+AD^{-1})\big)\\ &= \mathbb{I},\end{aligned}$$

so the equation is proved. Next we check that $D - AD^{-1}A$ is an M-matrix. Since A has all its entries nonpositive, and D nonnegative, the entries of $AD^{-1}A$ are nonnegative, so the off-diagonal entries of M' are nonpositive, as required. Regarding the diagonal entries, we compute

$$(D-AD^{-1}A)\mathbf{1} = \mathbf{d} - AD^{-1}\mathbf{a} \geq \mathbf{d} - A\mathbf{1} = \mathbf{d} - \mathbf{a} \geq \mathbf{0},$$

where both inequalities (here these are entrywise inequalities on vectors) use $\mathbf{a} \leq \mathbf{d}$ since by assumption the diagonal entries $\mathbf{d}$ are at least as large as the row sums $\mathbf{a}$. Moreover one of them must be strictly larger, so $\mathbf{d} - \mathbf{a} \neq 0$ and we are done. $\square$

We'll use the following notion of approximation.

Definition 15.3. For $\varepsilon > 0$ and symmetric matrices A and B we write $A \approx_\varepsilon B$ if

$$e^{-\varepsilon}A \leq B \leq e^{\varepsilon}A,[1]$$

where the inequalities are taken in the positive semidefinite order on matrices.

Given a diagonal matrix D and a symmetric matrix A, we say that (D, A) is an (α, β) pair if all entries of D are positive, all entries of A are nonnegative, and

$$-\beta D \leq A \leq \alpha D.$$

Lemma 15.4.

(1) *For any $\alpha < 1$, if (D, A) is an (α, α) pair, then $(D, AD^{-1}A)$ is an $(\alpha^2, 0)$ pair.*

(2) *For any $0 < \varepsilon \leq 1/3$, if (D, A) is a $(1-\delta, 0)$ pair, and $(\hat{D}, \hat{A})$ an (α, β) pair such that $D - A \approx_\varepsilon \hat{D} - \hat{A}$ and $D \approx_\varepsilon \hat{D}$ then $(\hat{D}, \hat{A})$ is an $(1-\delta e^{-2\varepsilon}, 3\varepsilon)$ pair.*

The first item in the lemma is what we really want: as long as $\alpha < 1$ it says that approximation quality *improves* through the transformation $M \to M'$ from Lemma 15.2. In particular, if (D, A) is say an $(1/4, 1/4)$ pair then we see that D^{-1} is a good approximation to $(D-A)^{-1}$, and so the equation $(D-A)^{-1}b = x$ is easy to solve.

Unfortunately we are not going to work exactly — at each step we'll apply a sparsification technique, which will avoid our matrices getting dense but will introduce approximation errors. The second item in the lemma shows that the transformation $M \to M'$ also behaves well with respect to approximations, as long as they are measured according to the "$\approx_\varepsilon$" relation.

Exercise 1. Prove Lemma 15.4.

[1] Here we use $e^{\pm\varepsilon}$ as a proxy for $(1 \pm \varepsilon)$. The advantage of this specific choice is that it gives us a notion of approximation which behaves very well with respect to taking products.

The algorithm. Given all that we've done, let's summarize how our candidate algorithm for solving $Mx = b$ is to proceed. First, we assume that M is an M-matrix, so it can be written as $M = M_0 = D_0 - A_0$ where (D_0, A_0) is an $(1-\kappa_0, 1-\kappa_0)$ pair, for some $0 < \kappa_0 < 1$. Note this will be guaranteed if $\|D^{-1/2}MD^{-1/2}\| \leq 1-\kappa_0$. In terms of graphs, this corresponds to setting κ_0 to be the second smallest eigenvalue of the normalized Laplacian, and we can also interpret it as the (inverse of the) condition number of the normalized adjacency matrix $\mathbb{I} - D^{-1/2}MD^{-1/2} = D^{-1/2}AD^{-1/2}$. As we'll see, the larger κ_0 the faster the algorithm will be.

Now, by Lemma 15.2 we know that the matrix $D_0 - A_0D_0^{-1}A_0$ is also an M-matrix, and by Lemma 15.4, $(D_0, A_0D_0^{-1}A_0)$ is an $((1-\kappa_0)^2, 0)$ pair. Moreover, Lemma 15.4 tells us that if we find (D_1, A_1) such that $D_0 - A_0D_0^{-1}A_0 \approx_\varepsilon D_1 - A_1$ and $D_0 \approx_\varepsilon D_1$ then (D_1, A_1) is also an $(1-\kappa_1, 3\varepsilon)$ pair for $\kappa_1 = 1-(1-\kappa_0)^2e^{-2\varepsilon} \approx 2\kappa_0$, provided ε is small enough.

Iterating this process, we construct a sequence (D_i, A_i) of $(1-\kappa_i, 3\varepsilon)$ pairs such that $\kappa_{i+1} \approx 2\kappa_i$. Repeating $O(\log \kappa_0^{-1})$ times we end up with a (say) $(1/4, 3\varepsilon)$ pair. For such a pair by (1) from Lemma 15.4 we have $0 \leq AD^{-1}A \leq (1/16)D$, and so $D - AD^{-1}A$ is well-approximated by just D. Chaining t applications of (15.1) together, we get

$$\begin{aligned}
(D_0 - A_0)^{-1} &= \frac{1}{2}\big(D_0^{-1} + (\mathbb{I} + D_0^{-1}A_0)(D_0 - A_0D_0^{-1}A_0)^{-1}(\mathbb{I} + A_0D_0^{-1})\big)\\
&\approx_\varepsilon \frac{1}{2}\big(D_0^{-1} + (\mathbb{I} + D_0^{-1}A_0)(D_1 - A_1)^{-1}(\mathbb{I} + A_0D_0^{-1})\big)\\
&\approx \qquad \cdots\\
&\approx \frac{1}{2}\Big(D_0^{-1} + (\mathbb{I} + D_0^{-1}A_0)\Big(\cdots(D_t - A_t)^{-1}\cdots\Big)(\mathbb{I} + A_0D_0^{-1}\Big)\\
&\approx \frac{1}{2}\Big(D_0^{-1} + (\mathbb{I} + D_0^{-1}A_0)\Big(\cdots(D_t^{-1})\cdots\Big)(\mathbb{I} + A_0D_0^{-1}\Big),
\end{aligned}$$

where the total error made in the approximations, using that with our notion of approximation errors add up, is order $t\varepsilon$, plus the small constant error made at the last step. To apply this matrix to b we have about $2t$ matrix-vector products to perform, all of which are easy! The D_i matrices are diagonal, so their inverses are easy to compute. And provided we managed to keep all the A_i matrices sparse, applying them is also easy. To obtain a small final error we need $\varepsilon \ll 1/t$. We're going to introduce a "sparsification" technique that lets us find A_i with $O(n/\varepsilon^2)$ nonzero entries, and thus the total cost of the algorithm is $O(n/\varepsilon^3)$, i.e. $O(n(\log(1/\kappa_0))^3)$. Note the huge improvement on the dependence on the condition number compared with the previous two algorithms!

There is one small issue we've kept under the rug. It seems that before even getting started on finding (D_1, A_1) that sparsify $(D_0, A_0D_0^{-1}A_0)$, we'd need to compute $A_0D_0^{-1}A_0$ in the first place...which could take too much time, at least $O(n^2)$ for squaring the matrix! Using some more ideas it is possible to avoid this, and never actually compute the matrix. We'll assume this can be done, so in the next lecture the goal will be, given an M-matrix $M = D - A$, find another M-matrix $\hat{M} = \hat{D} - \hat{A}$ such that $\hat{A}$ has $O(n/\varepsilon^2)$ entries and we have that both $D - A \approx_\varepsilon \hat{D} - \hat{A}$ and $D \approx_\varepsilon \hat{D}$, as required to apply item (2) in Lemma 15.4.

15.2 Sparsification

Suppose we are given a graph $G = (V, E)$, and we want to solve some kind of cut or partitioning problem. The running time will typically depend on the number of edges of the graph, which could be $O(n^2)$. Is there a way to "simplify" the graph by performing a pre-processing step, in order to efficiently produce a graph G' on a smaller number of edges but such that G' has roughly the same "cut structure" as G? This is the idea behind sparsification.

15.2.1 Edge sampling

A natural idea is to keep each edge with a certain probability p. Let's do a "back-of-the-envelope" calculation. If a cut $(S, \overline{S})$ involves c edges, and c' is the number of edges across that same partition in G', then on expectation $\mathbf{E}[c'] = pc$. Moreover, by the Chernoff bound,

$$\mathbf{Pr}\left(|pc - c'| \geq \varepsilon pc\right) \leq e^{-\varepsilon^2 pc/2}.$$

If we choose $p = \Omega(d \log n/(\varepsilon^2 c))$ for some d, then the probability drops to n^{-d}. It is possible to show (it is a theorem of Karger) that if the smallest cut in G has size c then the number of cuts of size αc is at most $n^{2\alpha}$. So if we are interested in preserving the size of all small cuts, say cuts of size at most Cn for some large C, we can choose d above large enough as a function of c and perform a union bound. This lets us take $p \approx \log n/(\varepsilon^2 n)$ and so we sparsify a dense graph using only $O(n \log n/\varepsilon^2)$ edges.

Unfortunately this only lets us preserve the size of the few cuts that are relatively small — for slightly larger cuts we'll have no guarantees. The problem with this method is that the concentration is too weak when a cut involves few edges. What is needed is a method to sample edges involved in fewer cuts with higher probability. (Think of the dumbbell graph: we don't want to miss that middle "bridge" edge!)

There is a way to achieve this which involves adding a weight w_e to each edge, such that $w_e \leq c$ where c is the size of the smallest cut in which the edge is involved. We sample edges with probability p/w_e and we assign them a weight of w_e. The expectation for the size of any cut is correct, as before. It is possible to show that this method does better — it provides guarantees that are more uniform across all cuts. But we're going to see how to do even better.

Sampling via effective resistances. We're going to use some form of importance sampling with a well-chosen set of weights. For any edge (a, b), let $r_{a,b} = (e_a - e_b)^T L^+(e_a - e_b)$, where L^+ is the pseudo-inverse of the Laplacian matrix (the inverse of L on its range) and e_a the indicator vector with a 1 in the a-th coordinate and 0 elsewhere. This quantity is called the *effective resistance* of edge (a, b), for good reason.

Anyone recall Ohm's law relating current through an edge to the difference of potential? If $I_{a\to b}$ units of current go from a to b, then we have $V(b) - V(a) = I_{a\to b}R$. Now, if we want to force potentials $V(a)$ at all vertices a of a weighted graph, with edge weights $w_{a,b}$

interpreted as inverse resistances (the larger the weight, the smaller the resistance), the amount of "exterior current" $I(a)$ that will have to be injected (or taken out, depending on sign) at node a is

$$\begin{aligned} I(a) &= \sum_{b\colon (a,b)\in E} w_{a,b}\big(V(a)-V(b)\big) \\ &= d_a V(a) - \sum_{b\colon (a,b)\in E} w_{a,b} V(b) \\ &= \big((D-A)V\big)_a . \end{aligned}$$

Therefore the exterior currents are given by $I = LV$. Inverting this, $V = L^+ I$. The *effective resistance* between a and b is defined as the equivalent amount of resistance that the graph induces if we inject one unit of current $a \to b$ and measure the resulting difference of potential between b and a:

$$r_{a,b} = V(b) - V(a) = (e_b - e_a)^T V = (e_b - e_a)^T L^+ (e_b - e_a),$$

where the $(e_b - e_a)^T$ represents measuring the difference of potential $V(b) - V(a)$, and the $(e_b - e_a)$ represents a vector of exterior currents where one unit flows from a to b.