CSC 2421H : Graphs, Matrices, and Optimization Lecture 9 : 19 November 2018

Approximate Gaussian Elimination for Laplacians

Lecturer: Sushant Sachdeva *Scribe: Yasaman Mahdaviyeh*

1 Introduction

Recall sparsifiers from last lecture. Given graph G we wanted to construct graph H such that $G \approx_\epsilon H$. We sampled edges of G with probability proportional to

$$p_e = w_e R_{eff}(e) = w_e(\mathbb{1}_u - \mathbb{1}_v)^\top \mathbf{L}_G^+(\mathbb{1}_u - \mathbb{1}_v),$$

where $e = (u, v)$. Computing $\boldsymbol{L}_G^+$ naively takes $O(n^3)$ operation. In this lecture we see that we can compute an approximation of the pseudoinverse of Laplacian much faster.

1.1 Laplacian Linear systems

We have the system $\boldsymbol{L}\mathbf{x} = \mathbf{b}$ where $\mathbf{b}$ is demands and we want to compute $\mathbf{x}$. This is a Laplacian linear system. It would be nice if we had an algorithm, let's call it LapInv(G, b), that computed $\boldsymbol{L}^+\mathbf{b} = \mathbf{x}$. This would have a wide range of applications, especially in scientific computing. There has been some work where $\mathbf{x}$ is approximated. Let $\mathbf{x}^\star = \boldsymbol{L}^+\mathbf{b}$ be the exact solution, and let $\mathbf{x}$ be output of the algorithm, then $\mathbf{x}^\star$ can be approximated in following sense

$$\left\|\mathbf{x} - \mathbf{x}^\star\right\|_{\boldsymbol{L}} \leq \epsilon \left\|\mathbf{x}^\star\right\|_{\boldsymbol{L}},$$

where $\|\mathbf{a}\|_L = \sqrt{\mathbf{a}^\top \boldsymbol{L}\mathbf{a}}$. All of the solvers discussed in next section give solutions in this norm. Note that this is only polynomially different from 2 norm because for all x such that $\mathbf{x}^\top \mathbb{1} = 0$

$$\lambda_2(\boldsymbol{L})\left\|\mathbf{x}\right\| \leq \left\|\mathbf{x}\right\|_{\boldsymbol{L}} \leq \lambda_{max}(\boldsymbol{L})\left\|\mathbf{x}\right\|$$

and

$$\frac{\lambda_2(\boldsymbol{L})}{\lambda_{max}(\boldsymbol{L})} \in O(n^3).$$

1.2 Relevant Work

Here m is number of edges, and n is number of vertices as usual.
[Spielman-Teng '04] [ST04] showed that we can do this approximation in O$(m \log^{O(1)} n \log \frac{1}{\epsilon})$.
[KMP' 11] [KMP11] O$(m \log^{1+o(n)} n \log \frac{1}{\epsilon})$
[Cohen et al. '14] [CKM$^+$14] $O(m\sqrt{\log n} \log \frac{1}{\epsilon})$
We will discuss following result which is much simpler and doesn't use graph theoretic constructions:
[Kyng-Sachdeva '16] [KS16] O$(m \log^3 n \log \frac{1}{\epsilon})$
First let's review guassian elimination.

1.3 Review of Gaussian Elimination

Suppose we want to solve this system of linear equations

$$\begin{aligned} 16x_1 - 4x_2 - 8x_3 - 4x_4 &= 16 \\ -4x_1 + 5x_2 - x_4 &= -6 \\ -8x_1 + 9x_3 - x4 &= 19 \\ -4x_1 - x_2 - x_3 + 7x_4 &= 11. \end{aligned}$$

One way of solving this system is to use the first equation to cancel out all x_1s in the rest and so on

$$\begin{aligned} 16x_1 - 4x_2 - 8x_3 - 4x_4 &= 16 \\ +4x_2 - 2x_3 - 2x_4 &= -2 \\ -2x_2 + 5x_3 - 3x4 &= -11 \\ 2x_2 - 3x_3 + 6x_4 &= 15. \end{aligned}$$

We could write the coefficients in a matrix and do the same thing. After one step, we can write the original matrix as

$$\begin{bmatrix} 16 & -4 & -8 & -4 \\ -4 & 5 & 0 & -1 \\ -8 & 0 & 9 & -1 \\ -4 & -1 & -1 & 7 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{-1}{4} \\ \frac{-1}{2} \\ \frac{-1}{4} \end{bmatrix} \begin{bmatrix} 16 & -4 & -8 & -4 \end{bmatrix} + \begin{bmatrix} 16 & -4 & -8 & -4 \\ 0 & 4 & -2 & -2 \\ 0 & -2 & 5 & -3 \\ 0 & 2 & -3 & 6 \end{bmatrix},$$

or equivalently, first row could be moved to the first term

$$\begin{bmatrix} 16 & -4 & -8 & -4 \\ -4 & 5 & 0 & -1 \\ -8 & 0 & 9 & -1 \\ -4 & -1 & -1 & 7 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{-1}{4} \\ \frac{-1}{2} \\ \frac{-1}{4} \end{bmatrix} \begin{bmatrix} 16 & -4 & -8 & -4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & -2 & -2 \\ 0 & -2 & 5 & -3 \\ 0 & 2 & -3 & 6 \end{bmatrix}.$$

Note that since the original matrix was symmetric, first term can be written as outer product of a rank 1 term with itself

$$\begin{bmatrix} 1 \\ \frac{-1}{4} \\ \frac{-1}{2} \\ \frac{-1}{4} \end{bmatrix} \begin{bmatrix} 16 & -4 & -8 & -4 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ -2 \\ -1 \end{bmatrix} \begin{bmatrix} 4 & -1 & -2 & -1 \end{bmatrix}.$$

2 Cholesky Factorization

Note that if we started with a matrix that was Laplacian (change 7 to 6), we would have the general form

$$\begin{bmatrix} d & -\mathbf{b}^\top \\ -\mathbf{b} & \mathbf{M} \end{bmatrix} = \begin{bmatrix} \sqrt{d} \\ \frac{-\mathbf{b}}{\sqrt{d}} \end{bmatrix} \begin{bmatrix} \sqrt{d} \\ \frac{-\mathbf{b}}{\sqrt{d}} \end{bmatrix}^\top + \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{M} - \frac{\mathbf{b}\mathbf{b}^\top}{d} \end{bmatrix}.$$

The submatrix $\mathbf{M} - \frac{\mathbf{b}\mathbf{b}^\top}{d}$ is $schur(\boldsymbol{L}_1, \{1\})$, where first vertex is eliminated. So as we saw in midterm question 5, Laplacian can be written as Laplacian = symmetric rank 1 + schur complement(Laplacian).

$$\boldsymbol{L} = \mathbf{v}_1\mathbf{v}_1{}^\top + \mathbf{S}^{(1)} = \mathbf{v}_1\mathbf{v}_1{}^\top + \mathbf{v}_2\mathbf{v}_2{}^\top + \mathbf{S}^{(2)} = ...$$

where $\mathbf{S}^{(i)}$ is schur complement of graph when first i vertices are eliminated. Note that $\mathbf{v}_2$ is zero in its first coordinate, similarly $\mathbf{v}_i$ has 0 in its first $i-1$ coordinates. Also, the order in which we eliminate vertices does not matter.

$$\boldsymbol{L} = \mathbf{v}_1\mathbf{v}_1{}^\top + \mathbf{v}_2\mathbf{v}_2{}^\top + ... + \mathbf{v}_n\mathbf{v}_n{}^\top$$

Writing it in matrix form we get

$$\boldsymbol{L} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & ... & \mathbf{v}_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1{}^\top \\ \mathbf{v}_2{}^\top \\ ... \end{bmatrix} = \mathbf{U}^\top\mathbf{U},$$

where $\mathbf{U}$ is an upper triangular matrix.
This symmetric factorization is called Cholesky factorization. Here we showed it for Laplacians, but in general it works for all positive semidefinite matrices. This is still expensive: $\mathrm{O}(n^3)$. In fact, it can be shown that for any Cholesky factorization (regardless of the order in which we eliminate vertices) of O(1) degree expanders we need $\Omega(n^3)$ steps. Why is Cholesky factorization useful ? Because it's easy to multiply by inverse of an upper triangular matrix

$$\boldsymbol{L}\mathbf{x} = \mathbf{b} \Leftrightarrow \mathbf{U}^\top\mathbf{U}\mathbf{x} = \mathbf{b} \Leftrightarrow \mathbf{x} = (\mathbf{U}^{-1})(\mathbf{U}^\top)^{-1}\mathbf{b}.$$

We can easily get $\mathbf{x}$ by forward substitution. The cost of this operation would be O(number of non zero entries in $\mathbf{U}$) = $\mathrm{O}(n^2)$. We cannot compute the upper triangular matrix $\mathbf{U}$ cheaply, so we will approximate it.

2.1 Introduction to Approximate Cholesky Factorization

Theorem 2.1 (Kyng-Sachdeva '16 [KS16]). *In $O(m\log^3 n)$ time, we can return upper triangular matrix $\mathbf{U}$ such that $\mathbf{U}^\top\mathbf{U} \approx_{1/3} \boldsymbol{L}$, that is $\frac{2}{3}\boldsymbol{L} \preceq \mathbf{U}^\top\mathbf{U} \preceq \frac{4}{3}\boldsymbol{L}$. Moreover, $\mathbf{U}$ has $O(m\log^3 n)$ non zero entries.*

Here for simplicity we fixed $\epsilon = 1/3$. There is a factor of $\frac{1}{\epsilon^2}$ in the running time if we hadn't fixed ϵ. We don't need high accuracy, we could always boost it. This will be presented in next section.
If we had an algorithm for fast approximate Cholesky factorization, then we could get a fast (approximate) Laplacian solver using following algorithm:

Corollary 2.2. *Following algorithm, called iterative refinement or Richardson iteration, returns $\mathbf{x}$ such that:*

$$\left\|\mathbf{x} - \boldsymbol{L}^+\mathbf{b}\right\|_{\boldsymbol{L}} \leq \epsilon \left\|\boldsymbol{L}^+\mathbf{b}\right\|_{\boldsymbol{L}}$$

for $t = O(\log\frac{1}{\epsilon})$.

Algorithm 1 Richardson Iteration

```
1: $\mathbf{x}^{(0)} \leftarrow 0$                                        ▷ initial guess
2: $\tilde{\mathbf{A}} \leftarrow (\mathbf{U}^\top\mathbf{U})^{-1}\boldsymbol{L}$
3: $(\mathbf{U}^\top\mathbf{U})^{-1}\mathbf{b} \leftarrow \tilde{\mathbf{b}}$    ▷ instead of solving $\boldsymbol{L}\mathbf{x} = \mathbf{b}$ , multiply both sides by $(\mathbf{U}^\top\mathbf{U})^{-1}$, and try to
   solve $(\mathbf{U}^\top\mathbf{U})^{-1}\boldsymbol{L}\mathbf{x} = (\mathbf{U}^\top\mathbf{U})^{-1}\mathbf{b}$
4: for $i \leftarrow 1 \dots t$ do
5:     $\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i-1)} - (\tilde{\mathbf{A}}\mathbf{x}^{(i-1)} - \tilde{\mathbf{b}})$        ▷ $\tilde{\mathbf{A}}$ can be multiplied cheaply
6: end for
```

Note that $\mathbf{U}^\top\mathbf{U}$ is not invertible, but has the same kernel as L (think pseudoinverse). For simplicity, we will hide those details here. Cost of each iteration = multiplication by L (O(m)) + multiplication by $\mathbf{U}^{-1}$ ($\text{O}(m\log^3 n)$) + multiplication by $(\mathbf{U}^\top)^{-1}$ $\text{O}(m\log^3 n)$

Proof of this corollary is left as an exercise, but here is a hint:
if $\mathbf{A}$ is symmetric and $\frac{2}{3}\mathbb{I} \preceq \mathbf{A} \preceq \frac{4}{3}\mathbb{I}$, then we saw in midterm:

$$\mathbf{A}^{-1} = \mathbb{I} + (\mathbb{I} - \mathbf{A}) + (\mathbb{I} - \mathbf{A})^2 + ...$$

We can get an approximation to A^{-1} if we only consider the first k terms of the sum. Specifically consider $k = 3\log\frac{1}{\epsilon}$.

2.2 Graph Interpretation of Cholesky Factorization

Consider one step of Cholesky factorization, where we are removing vertex 1:

$$\boldsymbol{L} = \frac{1}{d}\begin{bmatrix} d \\ -\mathbf{b} \end{bmatrix}\begin{bmatrix} d \\ -\mathbf{b} \end{bmatrix}^\top + \mathbf{S}^{(1)}$$

Let's think about what is happening to edges of the graph.
If (i,j) was such that $i, j \neq 1$, then (i,j) remains unchanged. Otherwise, we are removing all edges incident to vertex 1 and adding a clique to neighbours of vertex 1 (denoted by $N(1)$)

$$\mathbf{S}^{(i)} = \boldsymbol{L} - \frac{1}{d}\begin{bmatrix} d \\ -\mathbf{b} \end{bmatrix}\begin{bmatrix} d \\ -\mathbf{b} \end{bmatrix}^\top$$

$$\mathbf{S}^{(1)}_{i,j} = -w_{i,j}(\text{in L}) + \frac{1}{deg(1)}w(1,i)w(1,j).$$

Here is how we will approximate Cholesky Factorization: instead of adding a clique, we would add an approximation of a clique (we will sample a clique). The caveat is that we cannot have constant error in each iteration.

2.3 Sparse Approximate Cholesky Factorization

Consider following algorithm for Cholesky factorization

Algorithm 2 Exact Cholesky Factorization

```
1: for $k \leftarrow 1 \dots n$ do
2:     $\mathbf{c}_k \leftarrow \frac{1}{\sqrt{d_k}} \begin{bmatrix} d_k - \mathbf{b}_k \end{bmatrix}$          ▷ Record $k^{th}$ column
3:     Eliminate vertex k
4:
5:     Add a clique on $N(k)$.
6: end for
```

Now we can modify this to add an approximation of the clique, but to control the error, we cannot eliminate the vertices in arbitrary order, so we will pick one uniformly random at each iteration.

Algorithm 3 Sparse Approximate Cholesky Factorization

```
Replace each edge e by $\rho$ parallel edges each with weight $\frac{1}{\rho} w_e$          ▷ Preprocessing
for $k \leftarrow \dots n$ do
    Sample a vertex $\pi(k)$ uniformly random from remaining vertices

    Record $\mathbf{c}_k = \frac{1}{\sqrt{d_{\pi(k)}}} \begin{bmatrix} d_{\pi(k)} \\ -\mathbf{b}_{\pi(k)} \end{bmatrix}$

    Add Sample_Clique($N(k), k$) to $N(k)$ (where $N(k)$ denotes set of neighbours of vertex $k$)

end for
```

The preprocessing step is necessary because we want samples to have small norm so that we can use matrix concentration bounds. In the original graph norm of each sample is

$$\left\| w_e \boldsymbol{L}^{+/2} \boldsymbol{L}_e \boldsymbol{L}^{+/2} \right\| \leq 1$$

So scaling weight of each edge by $\frac{1}{\rho}$ will scale down norm of each sample to at most $\frac{1}{\rho}$. In next section we will see how the clique is sampled.

2.4 Sampling a Clique

Algorithm 4 Sample_Clique

```
procedure SAMPLE_CLIQUE($N(k), k$)
    for every edge $(k, a) \in N(k)$ do
        Sample $(k, b) \in N(k)$ with probability $\frac{w(k,b)}{w\_deg(k)}$

        Add the edge $(a, b)$ with weight $\frac{w(k,a)w(k,b)}{w(k,a)+w(k,b)}$ if $a \neq b$

    end for
return sampled edges
end procedure
```

Proposition 2.3. $\mathbb{E}$ *Laplacian of Sample_Clique $(N(k), k)$ = Laplacian of the weighted clique on neighbours of k in original graph.*

Proof. Let $\mathbf{Y}_a$ be laplacian of an added edge we get after we remove (k, a),

$$\mathbb{E}\,\mathbf{Y}_a = \sum_b \frac{w(k,b)}{deg(k)} \frac{w(k,a)w(k,b)}{w(k,a)+w(k,b)} \boldsymbol{L}_{a,b}$$

$$\sum_a \mathbb{E}\,\mathbf{Y}_a = \sum_{a,b,a<b} \frac{w(a,k)+w(b,k)}{deg(k)} \frac{w(a,k)w(b,k)}{w(k,a)+w(k,b)} \boldsymbol{L}_{a,b} = \sum_{a,b,a<b} \frac{w(k,a)+w(k,b)}{deg(k)} \boldsymbol{L}_{a,b} = Cl_k.$$

Note in the final term, we are adding Laplacians of all pairs of vertices adjacent to k, thus getting a clique. □

2.5 Expected Run Time

At each iteration, sampling vertex k costs $deg(k)$. So expected cost at step k is

$$\mathop{\mathbb{E}}_{\pi(k)} deg(k) = \frac{2m^{(k)}}{n-k+1} \leq \frac{2m\rho}{n-k+1},$$

where $m^{(k)}$ is number of edges at step k. The inequality holds because total number of edges is bounded by $m\rho$ and at each step number of edges does not increase. Therefore, total expected cost is $\leq 2m\rho(1+\frac{1}{2}+...+\frac{1}{n}) \leq 2m\rho \log n$.

References

[CKM+14] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly $m \log^{1/2} n$ time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 343–352, New York, NY, USA, 2014. ACM.

[KMP11] I. Koutis, G. L. Miller, and R. Peng. A nearly-$m \log n$ time solver for SDD linear systems. FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society.

[KS16] R. Kyng and S. Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, Oct 2016.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. STOC, pages 81–90, 2004.