Sixth Cargese Workshop on Combinatorial Optimization September 17, 2015

Lecture 3: Sparsifiers

Nick Harvey *University of British Columbia*

Abstract

In this last lecture we will discuss graph sparsification: approximating a graph by weighted subgraphs of itself. Sparsifications techniques have been used to design fast algorithms for combinatorial or linear algebraic problems, as a rounding technique in approximation algorithms, and have motivated startling results in pure mathematics.

1 Spectral Sparsifiers

Let $G = (V, E)$ be a graph with $V = \{1, \ldots, n\}$. Let e_i be the i^{th} standard basis vector. For an edge $e = \{u, v\} \in E$, let $\chi_e = e_u - e_v$, where u and v are ordered arbitrarily. The Laplacian of G is

$$L_G \;=\; \sum_{e \in E} \chi_e \chi_e^{\mathsf{T}}.$$

We say that a graph H on vertex set V *spectrally approximates* G if

$$(1-\epsilon) L_H \;\preceq\; L_G \;\preceq\; (1+\epsilon) L_H. \tag{1}$$

If H obtained from G by assigning non-negative weights to the edges, we call H a *spectral sparsifier* of G. The number of edges in H is defined to be the number of non-zero weights.

We will prove and discuss the following theorem.

Theorem 1. Every graph G has a spectral sparsifier with $O(n \log n/\epsilon^2)$ edges.

References: Spielman-Srivastava 2008.

1.1 Sparsifiers preserve cuts

For $U \subseteq V$, the set of edges in G with exactly one endpoint in U is denoted $\delta(U)$. A simple calculation shows that $|\delta(U)| = \mathbf{1}_U^{\mathsf{T}} L_G \mathbf{1}_U$, where $\mathbf{1}_U$ is the characteristic vector of the set U.

If H is a spectral sparsifier of G then (1) implies that

$$(1-\epsilon)\mathbf{1}_U^{\mathsf{T}} L_H \mathbf{1}_U \;\le\; \mathbf{1}_U L_G \mathbf{1}_U \;\le\; (1+\epsilon)\mathbf{1}_U L_H \mathbf{1}_U \qquad \forall U \subseteq V.$$

That is, the total weight in H of all edges in $\delta(U)$ lies in $[1-\epsilon, 1+\epsilon] \cdot |\delta(U)|$.

Application for fast algorithms. Consider any problem whose solution is determined by the cuts of G, e.g., the minimum s-t cut problem. Let $\mathcal{A}$ be an algorithm for this problem. Instead of running $\mathcal{A}$ on G, let us run it on the sparsifier H obtained from Theorem 1. The cut output by $\mathcal{A}$ is within $1+\epsilon$ of a minimum s-t cut in G. Running $\mathcal{A}$ on H instead of G is typically faster because H has fewer edges. Similar ideas can be used to give faster algorithms for the sparsest cut problem, the max cut problem, etc.

References: Benczur-Karger 2002, Fung et al. 2011.

2 Review

Recall that in the last lecture, we discussed Tropp's matrix Chernoff bound. Today we make a trivial extension by introducing the parameter R.

Theorem 2. Let $X_1, \ldots, X_k$ be independent random $n \times n$ symmetric matrices with $0 \preceq X_i \preceq R \cdot I$. Let $\mu_{\min} \cdot I \preceq \sum_i \mathrm{E}[X_i] \preceq \mu_{\max} \cdot I$. Then, for all $\epsilon \in [0,1]$,

$$\begin{aligned} \Pr\left[\lambda_{\max}(\textstyle\sum_{i=1}^k X_i) \geq (1+\epsilon)\mu_{\max}\right] &\leq n \cdot e^{-\epsilon^2 \mu_{\max}/3R} \\ \Pr\left[\lambda_{\min}(\textstyle\sum_{i=1}^k X_i) \leq (1-\epsilon)\mu_{\min}\right] &\leq n \cdot e^{-\epsilon^2 \mu_{\min}/2R}. \end{aligned}$$

3 Construction of Spectral Sparsifiers

We will prove Theorem 1 by an algorithm that randomly samples edges to create the weighted subgraph H. We will analyze the random matrix L_H using Theorem 2.

3.1 A Trick: Rescaling to Isotropic Position

One problem is that Theorem 2 only allows us to control the maximum and minimum eigenvalues of a random matrix, it does not directly allow us to establish matrix inequalities like (1). We will solve that problem by a simple rescaling trick.

The idea is to rescale χ_e by multiplying[1] it by $L_G^{-1/2}$. The resulting vectors are denoted

$$x_e \;=\; L_G^{-1/2}\chi_e.$$

They are said to be in "isotropic position" because

$$\sum_{e\in E} x_e x_e^{\mathsf{T}} \;=\; \sum_{e\in E} L_G^{-1/2}\chi_e\chi_e^{\mathsf{T}}L_G^{-1/2} \;=\; L_G^{-1/2}\Big(\sum_{e\in E}\chi_e\chi_e^{\mathsf{T}}\Big)L_G^{-1/2} \;=\; L_G^{-1/2}L_G L_G^{-1/2} \;=\; I.$$

Fact 3. Let $w : E \to \mathbb{R}_{\geq 0}$ and let H be the subgraph of G with edge weights w. Then (1) holds iff all eigenvalues of $\sum_{e\in E} w_e x_e x_e^{\mathsf{T}}$ lie in $[1-\epsilon, 1+\epsilon]$. Equivalently,

$$(1-\epsilon) \;\preceq\; \sum_{e\in E} w_e x_e x_e^{\mathsf{T}} \;\preceq\; (1+\epsilon)I \quad\Longleftrightarrow\quad (1-\epsilon)L_G \;\preceq\; \sum_{e\in E} w_e \chi_e\chi_e^{\mathsf{T}} \;\preceq\; (1+\epsilon)L_G.$$

Fact 3 follows from Fact 4, modulo the kernel caveat.

Fact 4. Let A and B be symmetric, $d \times d$ matrices and let C be any non-singular $d \times d$ matrix. Then $A \succeq B$ iff $CAC^{\mathsf{T}} \succeq CBC^{\mathsf{T}}$.

References: My notes on symmetric matrices.

3.2 The Proof

Due to Fact 3, our goal is to find weights $w : E \to \mathbb{R}_{\geq 0}$ such that all eigenvalues of $\sum_{e\in E} w_e x_e x_e^{\mathsf{T}}$ lie in $[1-\epsilon, 1+\epsilon]$. As mentioned above, we will do this by the random sampling process shown in Algorithm 1.

[1] Caveat: in this lecture we are going to ignore the fact that L_G is singular. This can be resolved by using the pseudoinverse rather than the inverse, and taking more care.

Algorithm 1: Sparsify a graph by sampling each edge ρ times with probability equal to its effective resistance. The sampled edges are weighted according to the reciprocal of their sampling probability.

Function `Sparsify(`G`)`**:**
  input: A graph $G = (V, E)$
  output: Edge weights $w : E \to \mathbb{R}_{\geq 0}$.
  Let $\rho = 6 \ln n/\epsilon^2$.
  Let p_e be the effective resistance across e, which is $\chi_e^\mathsf{T} L_G^{-1} \chi_e = x_e^\mathsf{T} x_e$.
  Initially $w \leftarrow 0$.
  Let $(Z_{i,e})_{i \in [\rho],\, e \in E}$ be mutually independent random variables in $\{0, 1\}$ with $\mathrm{E}[\, Z_{i,e} \,] = p_e$.
  for $i \leftarrow 1$ **to** ρ **do**
    foreach $e \in E$ **do**
      Increase w_e by $Z_{i,e}/\rho p_e$.
  Return w

Remark. Sampling edges with probability proportional to their effective resistances is a well-studied approach in the statistics literature, where it is known as *leverage score sampling*.

References: Mahoney 2011.

Claim 5. The expected number of non-zero edge weights is $6n \log n/\epsilon^2$.

Proof: The expected number of non-zero edge weights is at most

$$\sum_{i=1}^{\rho} \sum_{e \in E} p_e \;=\; \rho \cdot \sum_{e \in E} x_e^\mathsf{T} x_e \;\overset{(*)}{=}\; \rho \cdot \mathrm{tr} \sum_{e \in E} x_e x_e^\mathsf{T} \;=\; \rho \cdot \mathrm{tr}\, I \;=\; (6 \ln n/\epsilon^2) \cdot n$$

In $(*)$ we have used the cyclic property of trace. □

Claim 6. Let $X = \sum_{e \in E} w_e x_e x_e^\mathsf{T}$. With high probability, all eigenvalues of X lie in $[1 - \epsilon, 1 + \epsilon]$. Consequently, the graph H with weights w is a sparsifier of G with high probability.

Proof: The matrix X that we wish to analyze is constructed by summing a contribution from each iteration of the algorithm. The random matrix contributed in iteration (i, e) is

$$X_{i,e} \;=\; Z_{i,e} \cdot x_e x_e^\mathsf{T}/\rho p_e \;=\; Z_{i,e} \cdot \left(\frac{x_e x_e^\mathsf{T}}{x_e^\mathsf{T} x_e}\right) \cdot \frac{1}{\rho}. \tag{2}$$

Thus $X = \sum_{i,e} X_{i,e}$.

Our first observation is that

$$\mathrm{E}[\, X \,] \;=\; \sum_{i,e} \mathrm{E}[\, X_{i,e} \,] \;=\; \rho \sum_{e} \underbrace{\mathrm{E}[\, Z_{i,e} \,]}_{=p_e} \cdot (x_e x_e^\mathsf{T}/\rho p_e) \;=\; \sum_{e} x_e x_e^\mathsf{T} \;=\; I.$$

So all eigenvalues of $\mathrm{E}[\, X \,]$ are 1.

Next we claim that all eigenvalues of $X_{i,e}$ are in $\{0, 1/\rho\}$. To see this, consider (2) and note that the matrix $\frac{x_e x_e^\mathsf{T}}{x_e^\mathsf{T} x_e}$ has all eigenvalues in $\{0, 1\}$. So $X_{i,e}$ has eigenvalues in $\{0, 1/\rho\}$ when $Z_{i,e} = 1$; otherwise all eigenvalues are 0.

We wish to show that, with high probability, all eigenvalues of X are in $[1 - \epsilon, 1 + \epsilon]$. To do so, we

apply Theorem 2 with $R = 1/\rho$ and $\mu_{\max} = \mu_{\min} = 1$, obtaining

$$\begin{aligned}
\Pr\left[\lambda_{\max}(\textstyle\sum_{i,e} X_{i,e}) \geq (1+\epsilon)\right] &\leq n \cdot e^{-\epsilon^2/3R} = n \cdot e^{-2\ln n} = 1/n \\
\Pr\left[\lambda_{\min}(\textstyle\sum_{i,e} X_{i,e}) \leq (1-\epsilon)\right] &\leq n \cdot e^{-\epsilon^2/2R} = n \cdot e^{-3\ln n} \leq 1/n.
\end{aligned}$$

By a union bound, the probability that the eigenvalues of X don't lie in $[1-\epsilon, 1+\epsilon]$ is at most $2/n$. □

Combining Claim 5 and Claim 6 proves Theorem 1.

4 The Koutis-Miller-Peng Algorithm

The fast Laplacian solver of Spielman-Teng [8] relied on many components, including a spectral sparsification steps (followed by an "ultra-sparsification" step). We have just seen an algorithm for computing very good sparsifiers, but unfortutately it requires computing effective resistances, for which we need a fast Laplacian solver. This is a chicken-and-egg problem!

Koutis-Miller-Peng [5] presented a fast Laplacian solver that is substantially more comprehensible than the Spielman-Teng algorithm. The main idea of KMP is a clever resolution of the chicken-and-egg problem. Basically, they use low-stretch trees to give a very rough estimate for the effective resistances, then use those estimates to do random sampling like in Algorithm 1. This gives a sparsifier that doesn't near-linear number of edges, but the number of edges does decrease by a polylogarithmic factor. The KMP algorithm then recursively solving the Laplacian system on the sparsifier, and uses that to solve the original Laplacian system.

So now you have seen the two key ingredients of the KMP solver: low-stretch trees and spectral sparsifiers. You are now fully equipped to read the KMP paper — all that remains are some technical details on how the recursion works!

References: Koutis-Miller-Peng 2010, Vishnoi 2013

5 Beyond Independent Sampling

The previous theorem is as good (fewest edges) as one could hope when independently sampling edges of a graph. For example, a random sparsifier of complete graph is the same as a $G(n,p)$ graph with $p = 2\rho/n$, which is likely disconnected if $\rho = o(\log n)$, and therefore does not meet the spectral guarantee.

Can dependent sampling or deterministic algorithms give better results than independent sampling? For example, sampling $\Theta(1/\epsilon^2)$ *perfect matchings* in the complete graph will give a near-Ramanujan graph, which is a spectral sparsifier.

5.1 Linear-size sparsifiers

Batson, Spielman and Srivastava [1] showed that any undirected graph has a spectral sparsifier with $O(n/\epsilon^2)$ edges. Rather than use dependent sampling, they give a deterministic, polynomial-time algorithm.

To motivate their approach, let us consider the derandomization of Chernoff bounds. There is a technique called the method of pessimistic estimators (based on the method of conditional expectations) that allows one to deterministically choose values for the "random variables" one-by-one so that the desired bounds will be satisfied. The main idea is to use some "potential function" based on the

algorithm's previous choices that will indicate whether the algorithm's next choice is good or not. The potential function is used is basically the function $e^{-\theta t} \prod_i \mathrm{E}\left[e^{\theta X_i} \right]$ that appeared in the proof of the Chernoff bound. The resulting algorithm ends up being essentially the multiplicative-weight update method. This approach can be extended to work for the Matrix Chernoff Bound.

References: my 2015 lecture notes, de Carli Silva-Harvey-Sato 2011, Arora-Hazan-Kale 2012.

Using these exponential functions, one cannot hope to do better than the Chernoff bound. The BSS algorithm is based the function $1/x$ instead of e^x; because $1/x$ has a pole it acts as a "barrier function". They use these barriers to keep the eigenvalues in a tight interval as they choose edges to add to the graph.

Allen-Zhu et al. put these ideas in a more general context and gave a faster algorithm based on connections to regret-minimization algorithms. In particular, they show that the $1/x$ function naturally arises from regularization ideas. Lee and Sun (2015) gave a nearly-linear time algorithm by a careful batching strategy. Both of these use fast Laplacian solvers as a subroutine.

References: Allen-Zhu, Liao and Orecchia 2015, Lee-Sun 2015.

References

[1] J. Batson, D. A. Spielman, and N. Srivastava. Twice-Ramanujan sparsifiers. In *STOC*, 2009.

[2] A. A. Benczúr and D. R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs, 2002. `http://arxiv.org/abs/cs/0207078`.

[3] M. de Carli Silva, N. J. A. Harvey, and C. Sato. Sparse sums of positive semi-definite matrices. arXiv:1107.0088.

[4] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, 2011.

[5] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving sdd linear systems. In *FOCS*, 2010.

[6] M. W. Mahoney. *Randomized Algorithms for Matrices and Data*, volume 3 of *Foundations and Trends in Machine Learning*. now publishers, 2011.

[7] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *STOC*, 2008.

[8] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90, 2004.

[9] N. K. Vishnoi. *Lx=b*, volume 8 of *Foundations and Trends in Theoretical Computer Science*. now publishers, 2013.