# CS 466 / 666 : Algorithm Design and Analysis . Spring 2018 . Waterloo.

## Lecture 16: Review and preview

We do a quick review of what we have learnt, and then a quick preview of what we have not covered, to complete the story that was told in the first lecture of this course.

---

### Review

In the beginning, we plan to do randomized algorithms, linear algebra, and linear programming.
In the end, we have only completed the part on randomized algorithms, slightly more than half of linear algebra, and one lecture on linear programming (but the part we are missing on linear programming does not affect our story too much).

Let's quickly review what we have learnt, with an emphasis on what I would like you to know well.
There are five components in randomized algorithms.

① Concentration inequalities : We have learnt the basic inequalities (Markov, Chebyshev, Chernoff).
These are the most fundamental tools in analyzing randomized algorithms and I really would like to see that you know them well.
It is not so important to remember the specific applications that we have seen well (e.g. graph sparsification).
Rather, I would like to see that you can apply these inequalities in new but relatively simple settings.

② Balls and bins, k-wise independence, hashing : Able to bound some basic quantities in the balls-and-bins model, and see that these simple random processes (e.g. coupon collector) give us useful bounds for many problems.
Understand the concept of k-wise independence and what is hashing.
The specific applications in data streaming are not that important, but rather the technical ideas therein (e.g. the use of Chebyshev's inequality).

③ Polynomial identity testing : Understand the Schwartz-Zippel lemma, and that the polynomial identity testing can be used to solve different problems.
The specific applications (network coding, parallel matching) are not that important.

④ Probabilistic methods : How to use the first and the second moment method to prove the existence of

combinatorial objects, basically these methods are just Markov and Chebyshev inequalities.

Basic applications of the local lemma. Moser's algorithm is really interesting, but we have already done it in the homework (so hopefully you won't forget).

⑤ Random walks: Understand the fundamental theorem and its implications (e.g. hitting time).

The specific applications are not that important (pagerank, bipartite matching) and we have already covered them in homework problems.

This topic leads us to the study of spectral analysis.

The next part is about spectral graph theory and linear algebra.

This is probably more difficult for most students, so we will just focus on the basics.

- Spectral graph theory: Understand the connections between the eigenvalues and combinatorial properties (e.g. $\lambda_2$ and connectedness, $\alpha_n$ and bipartiteness, $\lambda_1$ and degrees, $\lambda_k$ and number of components).

  Know the Laplacian matrix and its basic properties (PSD and quadratic form).

  Understand the Rayleigh quotient characterization of eigenvalues.

- Cheeger's inequality: It is an important and cool result, but we'll skip it for the exam.

  Just understand the easy direction is enough.

- Mixing time: Understand the spectral analysis using eigen-decomposition to show that the first eigenvector will stand out after repeated application of the same matrix.

- Electrical networks: Understand that a Laplacian system of equations is an electrical flow problem, but don't worry about the pseudo-inverse of the Laplacian.

  Understand the concept of effective resistance and some basic bounds (e.g. disjoint short paths), but don't worry about the proof of the Thompson's principle and the triangle inequality.

  Understand the connection between commute time / cover time to effective resistance.

Finally, I would like you to understand the multiplicative weight update method as it is useful in different areas. The application to solving LP is nice to know, but we have covered it in HW5.

In general, focus on the basic concepts and techniques, rather than the more advanced proofs and applications.

I hope you will be technically solid and feel comfortable with probabilities and okay with linear algebra.

These techniques will become more and more important in computer science (e.g. machine learning).

---

## Preview

I would really like to give you some ideas about three interesting results that we planned to study, that also point to the current research in algorithm design.

① Solving maximum flow using electrical flow and multiplicative update method.

This opens up an exciting research direction in using continuous optimization for combinatorial problems.

② Graph sparsification by effective resistance.

This shows that the correct way to look at this combinatorial problem is through linear algebra.

This result also leads to further breakthroughs in CS and math.

③ Near linear time algorithm for solving Laplacian system of equations (i.e. electrical flow).

This is the engine behind the revolution mentioned above in ①.

---

## Maximum Flow in Undirected Graphs

In this problem, we are given an undirected graph where each edge $e$ has a capacity $c_e$, a source vertex $s$, and a sink vertex $t$, and the objective is to find a maximum flow subjected to the capacity constraints, where a flow has to satisfy the flow conservation constraints.

More formally, we have two variables $f_{uv}$ and $f_{vu}$ for each edge $uv$, and the problem is to:

$$\max \sum_{e \in \delta^{out}(s)} f_e$$ (where $\delta^{out}(s)$ is the set of directed edges going out from $s$)

$$\sum_{e \in \delta^{out}(v)} f_e = \sum_{e \in \delta^{in}(v)} f_e \quad \text{for all } v \in V - \{s,t\}$$ (where $\delta^{in}(v)$ is the set of incoming edges to $v$)

$$f_e \le c_e \quad \text{for all } e$$

$$f_e \ge 0 \quad \text{for all } e$$

For simplicity, we assume $c_e = 1$ for all edge. So the optimal value of the linear program is in $[0, m]$ where $m$ is the number of edges. As mentioned before, we can reduce the optimization problem to $O(\log m)$ decision problems, by doing binary search on the objective value and replace the objective function by the constraint $\sum_{e \in \delta^{out}(s)} f_e = k$.

## Multiplicative update method

## Multiplicative update method

As in homework 5, we will apply the multiplicative weight update method to solve the problem.

We think of the objective constraint ( $\sum_{e \in \delta_{out}(s)} f_e = k$ ), the non-negative constraints ( $f_e \geq 0 \quad \forall e \in E$ ), and the flow conservation constraint ( $\sum_{e \in \delta_{in}(v)} f_e = \sum_{e \in \delta_{out}(v)} f_e \quad \forall v \in V - \{s,t\}$ ) as "easy" constraints, and we will always satisfy them.

The capacity constraints ( $f_e \leq 1 \quad \forall e \in E$ ) are the "hard" constraints, and we will use the multiplicative weight update method to deal with them.

By the analysis of the multiplicative weight update method, we can find an almost feasible solution ( $f_e \leq 1+\varepsilon \quad \forall e \in E$ ) if we could solve $O(\frac{\rho \ln n}{\varepsilon^2})$ subproblems of the following form:

(0) $f$ satisfies the objective constraint, the non-negative constraints, and the flow conservation constraints.

(1) $\sum_e w_e f_e \leq (1+\varepsilon) \sum_e w_e$. In the homework problem, this is $\sum_e w_e f_e \leq \sum_e w_e$, but it is easy to check that it is okay to respond with an approximate feasible solution, as we do approximation anyway.

(2) $f_e \leq \rho$, where $\rho$ is the width parameter.

If these subproblems can be solved (for different $w_e$), then the average solution would be an almost feasible solution, and by scaling it would satisfy all capacity with objective $k(1-\varepsilon)$.

**Remark:** In the multiplicative update method in L15, the convergence rate is $O(\frac{\rho^2 \ln n}{\varepsilon^2})$.

That analysis is for the general case where the outcome is in $[-1,+1]$.

It is not difficult to show that if the outcome is in $[0,1]$, then the convergence rate is $O(\frac{\rho \ln n}{\varepsilon^2})$.

---

## Electrical flow

In the homework problem, we use a shortest path (on the weight $w_e$) to solve the subproblem with width $k$.

Here we show that the electric flow algorithm can be used to solve the subproblem with the following properties:

(0) It returns $f$ that satisfies flow conservation constraints, non-negative constraints, and objective value constraint.

(1) When the maximum flow problem is feasible, the oracle can always return $f$ satisfying $\sum_e w_e f_e \leq (1+\varepsilon) \sum_e w_e$.

(2) When the maximum flow problem is feasible, the oracle can always return $f$ with $f_e \leq O(\sqrt{m})$.

(3) The oracle can be implemented in $\tilde{O}(m)$ time.

Assuming this can be done, we can find an $(1-\varepsilon)$-approximate solution in $O(\frac{\sqrt{m} \ln n}{\varepsilon^2})$ iterations,

Assuming this can be done, we can find an $(1-\varepsilon)$-approximate solution in $O(\frac{\sqrt{m}\ln n}{\varepsilon^2})$ iterations, and the total running time is $\tilde{O}(m^{3/2}/\varepsilon^2)$.

It remains to construct the oracle with the above guarantees. As we said earlier we use electric flow.

**Oracle** The oracle can be obtained by setting $r_e = w_e + \frac{\varepsilon W}{m}$ where $W = \sum_e w_e$, and compute the electric flow that sends $k$ units of electric flow from $s$ to $t$ with $r_e$ as the resistance of edge $e$.

This is the oracle. So the whole algorithm is very simple. In each iteration, $w_e$ is updated by $w_e^{t+1} = w_e^t(1+\frac{\varepsilon}{\rho}f_e^t)$ and update $r_e$ accordingly, then compute the electric flow. So, if the flow on an edge is over its capacity, we will increase its resistance based on its violation to decrease the future flow on this edge. Taking the average over all the electric flow is a good approximation.

**Analysis**

We check the four properties one by one.

⓪ It is clear by construction that the flow conservation constraints, the non-negativity constraints, and the objective value constraint are all satisfied.

① If the maximum flow problem is feasible, then there is a flow of $k$ units from $s$ to $t$, without violating the capacity constraints (so $f_e \le 1 \ \forall e \in E$).

The total energy of this flow is at most $\sum_e \tilde{f}_e^2 r_e \le \sum_e r_e = \sum_e (w_e + \frac{\varepsilon W}{m}) = (1+\varepsilon)W$.

Since electric flow $f$ minimizes the energy, we must have $\sum_e f_e^2 r_e \le (1+\varepsilon)W$, or otherwise we can conclude that the maximum flow problem is infeasible

We would like to bound $\sum_e w_e f_e$, and we would use Cauchy-Schwarz.

$\left(\sum_e w_e f_e\right)^2 \le \left(\sum_e w_e f_e^2\right)\left(\sum_e w_e\right) \le \left(\sum_e r_e f_e^2\right)\left(\sum_e w_e\right) \le (1+\varepsilon)W^2$, and so we have $\sum_e w_e f_e \le (1+\varepsilon)\sum_e w_e$, as required.

② Clearly the energy on one edge cannot be more than the total energy,

so $\frac{f_e^2 \varepsilon W}{m} \le f_e^2 r_e \le (1+\varepsilon)W \Rightarrow f_e \le \sqrt{\frac{(1+\varepsilon)}{\varepsilon} m} = O(\sqrt{m})$, as required.

③ As we saw in L14, computing electric flow is the same as solving Laplacian systems, which we will discuss how this can be done in $\tilde{O}(m)$ time.

This completes the $\tilde{O}(m^{1.5})$ time algorithm to give an $(1-\varepsilon)$-approximation of the maximum flow problem in undirected graphs, matching the best known combinatorial method.

This approach can be improved to $\tilde{O}(m^{4/3}/\varepsilon^2)$ time.

Now we know $\tilde{O}(m/\varepsilon^2)$ time algorithm for undirected max flow, not by multiplicative update but some other continuous optimization method, and many more new algorithms along this line.

---

## Spectral sparsification

Recall that a graph $H$ is a $(1\pm\varepsilon)$-cut approximator of $G$ if $(1-\varepsilon)w_G(\delta(S)) \le w_H(\delta(S)) \le (1+\varepsilon)w_G(\delta(S))$ for all $S \subseteq V$, where $w_G(\delta(S))$ is the total weight of the edges crossing $S$.

We mentioned the result by Benczur and Karger that for any $G$, there exists a $(1\pm\varepsilon)$-cut approximator with $O(n\log n/\varepsilon^2)$ edges.

Today we will prove a spectral generalization of this result.

### Spectral approximator

We say a graph $H$ is a $(1\pm\varepsilon)$-spectral approximator of $G$ if $(1-\varepsilon)L_G \preceq L_H \preceq (1+\varepsilon)L_G$, or equivalently $(1-\varepsilon)x^T L_G x \le x^T L_H x \le (1+\varepsilon)x^T L_G x \quad \forall x \in \mathbb{R}^n$ where $n$ is the number of vertices.

**Claim** A $(1\pm\varepsilon)$-spectral approximator is a $(1\pm\varepsilon)$-cut approximator.

**Proof** For $S \subseteq V$, let $x_S \in \mathbb{R}^n$ be the vector with $x_S(i)=1$ if $i \in S$ and zero otherwise.

Then, $x_S^T L_G x_S = \sum_{ij\in E} w(ij)\,(x_S(i)-x_S(j))^2 = w_G(\delta(S))$ and similarly $x_S^T L_H x_S = w_H(\delta(S))$.

Since $H$ is a $(1\pm\varepsilon)$-spectral approximator of $G$, we have

$(1-\varepsilon)x_S^T L_G x_S \le x_S^T L_H x_S \le (1+\varepsilon)x_S^T L_G x_S \quad \forall S \subseteq V$ and thus $(1-\varepsilon)w_G(\delta(S)) \le w_H(\delta(S)) \le (1+\varepsilon)w_G(\delta(S)) \quad \forall S \subseteq V$. □

The following theorem by Spielman and Srivastava thus generalizes the result of Benczur and Karger.

**Theorem** Any graph has a $(1\pm\varepsilon)$-spectral approximator with $O(n\log n/\varepsilon^2)$ edges.

---

## Reduction

The spectral sparsification result can be reduced to the following purely linear algebraic result.

**Theorem** Suppose $v_1,\dots,v_m \in \mathbb{R}^n$ are given with $\sum_{i=1}^m v_i v_i^T = I_n$.

There exist scalars $s_1,\dots,s_m$ with at most $O(n\log n/\varepsilon^2)$ non-zeros such that

$$(1-\varepsilon)I_n \le \sum_{i=1}^m s_i v_i v_i^T \le (1+\varepsilon)I_n.$$

We sketch the proof of the reduction of the spectral sparsification result to the above result.

The idea is to apply a linear transformation so that the Laplacian matrix becomes the identity matrix.

Let $M$ be a positive semidefinite matrix with eigen-decomposition $M = \sum_{i=1}^{n} \lambda_i u_i u_i^T$.

The pseudo-inverse of $M$ is defined as $M^{\dagger} = \sum_{i:\lambda_i > 0} \frac{1}{\lambda_i} u_i u_i^T$, and $M^{\dagger/2} = \sum_{i:\lambda_i>0} \frac{1}{\sqrt{\lambda_i}} u_i u_i^T$.

Given $L_G = \sum_{e \in E} L_e = \sum_{e \in E} b_e b_e^T$, we consider $I = L_G^{\dagger/2} L_G L_G^{\dagger/2} = \sum_{e \in E} (L_G^{\dagger/2} b_e)(b_e^T L_G^{\dagger/2}) = \sum_{e \in E} v_e v_e^T$,

where we define $v_e = L_G^{\dagger/2} b_e \quad \forall e \in E$.

Apply the above theorem gives us $s_e$ with at most $O(n \log n / \varepsilon^2)$ non-zeros so that

$$(1-\varepsilon) I \leq \sum_{e \in E} s_e v_e v_e^T \leq (1+\varepsilon) I .$$

Now, multipling $L_G^{1/2}$ on the left and right gives us $(1-\varepsilon) L_G \leq \sum_{e \in E} s_e b_e b_e^T \leq (1+\varepsilon) L_G$, so by scaling the weight of each edge by a factor of $s_e$, we get our spectral sparsifier.

The above "proof" is not precise as we are dealing with the pseudo-inverse (but not the inverse), but the missing details are rather routine and is not the important part of the proof, and so omitted.

---

## Sampling algorithm

Now, our focus is to prove the linear algebraic result, by random sampling.

First, we get some intuition about the condition $\sum_{i=1}^{m} v_i v_i^T = I_n$.

When $m = n$, then $v_1, \ldots, v_n$ must be an orthonormal basis.

When $m > n$, we can also think of it as an "overcomplete" basis, as we can write any $x \in \mathbb{R}^n$ as

$$x = I_n x = \left(\sum_{i=1}^{m} v_i v_i^T\right) x = \sum_{i=1}^{m} \langle x, v_i \rangle v_i .$$

Similarly, for any unit vector $y \in \mathbb{R}^n$, we have $1 = y^T y = y^T I y = y^T \left(\sum_{i=1}^{m} v_i v_i^T\right) y = \sum_{i=1}^{m} y^T v_i v_i^T y = \sum_{i=1}^{m} \langle v_i, y \rangle^2$.

Intuitively, the vectors are "evenly spread out", so that the projection of any direction $y$ to these vectors are the same.

Idea: Given $\sum_{i=1}^{m} v_i v_i^T = I_n$, we would like to find a small subset of vectors $S \subseteq \{1, \ldots, m\}$ and some scaling factors so that $\sum_{i \in S} s_i v_i v_i^T \approx I_n$.

So, the subsets should still be "evenly spread out", with contributions in each direction about the same.

As in the graph sparsification case, uniform sampling won't work. For example, if some $v_j$ has $\|v_j\| = 1$, then we must include $v_j$ in the solution, as otherwise that direction will not be covered in

the solution and so it won't be a spectral sparsifier. The analogy in the graph sparsification result is that a cut edge must be included in any sparsifier.

So, as in the graph sparsification case, we need to do non-uniform sampling (if we do random sampling). The idea is similar: for longer vectors, the sampling probability is higher; for shorter vectors, we can be more aggressive in setting the sampling probability to be smaller, and when we choose them, we reweight the vector so that it has the correct expected value.

More concretely, we sample each vector $v_i$ with probability $\|v_i\|_2^2$, and if it is chosen, we set the scaler $s_i = \frac{1}{\|v_i\|_2^2}$, so that $E[s_i v_i v_i^T] = \frac{v_i v_i^T}{\|v_i\|_2^2} \cdot \Pr(v_i \text{ is chosen}) = \frac{v_i v_i^T}{\|v_i\|_2^2} \cdot \|v_i\|_2^2 = v_i v_i^T$.

## Algorithm

The actual algorithm is basically the same as described above, but we need to repeat this experiment $C = \Theta(\log n)$ times and take the average, so that we can prove concentration.

- Initially, $F \leftarrow \phi$, $s \leftarrow 0$, $C = \frac{6 \log n}{\varepsilon^2}$
- For $1 \le t \le C$ do

  For each $e \in E$, with probability $p_i = \|v_i\|_2^2$, update $F \leftarrow F \cup \{i\}$ and $s_i \leftarrow s_i + \frac{1}{C p_e}$.
- Return $\sum_{i \in F} s_i v_i v_i^T$ as our spectral approximator.

## Analysis

There are two steps in the analysis.

One is to show that there are $O(n \log n / \varepsilon^2)$ non-zeros scalars, i.e. $|F| = O(n \log n / \varepsilon^2)$.

Another is to show that the returned solution is a $(1 \pm \varepsilon)$-spectral sparsifier.

We first bound the number of non-zeros scalars.

**Claim** With probability at least 0.9, $|F| = O(n \log n / \varepsilon^2)$.

**Proof** The expected value is $E[|F|] = \sum_{i=1}^m \Pr(\text{vector } i \text{ is in } F) = \sum_{i=1}^m (1 - (1-p_i)^C) \le \sum_{i=1}^m (1 - (1 - Cp_i)) = C \cdot \sum_{i=1}^m p_i$,

which can also be seen by a union bound.

Note that $\sum_{i=1}^m p_i = \sum_{i=1}^m \|v_i\|_2^2 = \sum_{i=1}^m v_i^T v_i = \sum_{i=1}^m \mathrm{tr}(v_i^T v_i) = \sum_{i=1}^m \mathrm{tr}(v_i v_i^T) = \mathrm{tr}\left(\sum_{i=1}^m v_i v_i^T\right) = \mathrm{tr}(I_n) = n$, where

$\mathrm{tr}(A) = \sum_j A_{jj}$ and we use the fact that $\mathrm{tr}(AB) = \mathrm{tr}(BA)$ (or directly check that $v^T v = \mathrm{tr}(v v^T)$).

Therefore, $E[|F|] \le C \sum_{i=1}^m p_i = Cn = 6 n \log n / \varepsilon^2$. The result follows from Markov's inequality. □

## Matrix Chernoff bound

There is an elegant generalization of the Chernoff-Hoeffding bound to the matrix setting.

**Theorem** (Tropp) Let $X_1, \ldots, X_k$ be independent, $n \times n$ symmetric matrices with $0 \preceq X_i \preceq RI$.

Let $\mu_{min} I \preceq \sum_{i=1}^{k} E[X_i] \preceq \mu_{max} I$. For any $\varepsilon \in [0,1]$,

- $Pr\left( \lambda_{max}\left( \sum_{i=1}^{k} X_i \right) \geq (1+\varepsilon)\mu_{max} \right) \leq n e^{-\frac{\varepsilon^2 \cdot \mu_{max}}{3R}}$
- $Pr\left( \lambda_{min}\left( \sum_{i=1}^{k} X_i \right) \leq (1-\varepsilon)\mu_{min} \right) \leq n e^{-\frac{\varepsilon^2 \cdot \mu_{min}}{2R}}$.

Note that it is almost an exact analog of the Chernoff-Hoeffing bound in the scalar case, by using the maximum eigenvalue and minimum eigenvalue to measure the "size" of a matrix.

It says that if we consider the sum of independent random matrices, where each matrix is not too "big/influential", then the sum is concentrated around the expectation in terms of the eigenvalues.

## Concentration

The proof that our solution is a $(1\pm\varepsilon)$-spectral sparsifier is a direct application of the matrix Chernoff bound.

The random variables are $X_{i,t} = \begin{cases} \frac{v_i v_i^T}{C p_i} & \text{with probability } p_i = \|v_i\|_2^2, \text{ for vector } i \text{ in iteration } t. \\ 0 & \text{otherwise} \end{cases}$

Note that the output of the algorithm is $S := \sum_{t=1}^{C} \sum_{i=1}^{m} X_{i,t}$.

As discussed before, $E[S] = \sum_{t=1}^{C} \sum_{i=1}^{m} E[X_{i,t}] = \sum_{t=1}^{C} \sum_{i=1}^{m} \frac{v_i v_i^T}{C p_i} \cdot p_i = \sum_{t=1}^{C} \sum_{i=1}^{m} \frac{v_i v_i^T}{C} = \sum_{i=1}^{m} v_i v_i^T = I$.

So, the expected value is correct, with $\mu_{max} = \mu_{min} = 1$ in this problem.

To apply the matrix Chernoff bound, we just need to find a bound for $R$ so that $X_{i,t} \preceq RI$.

Note that $X_{i,t} = \frac{v_i v_i^T}{C p_i} = \frac{v_i v_i^T}{C \|v_i\|_2^2} = \frac{1}{C}\left(\frac{v_i}{\|v_i\|}\right)\left(\frac{v_i}{\|v_i\|}\right)^T$. This is a rank one matrix of a unit vector, and so the maximum eigenvalue is just $\frac{1}{C}$ (with the only eigenvector being $\frac{v_i}{\|v_i\|}$). So, $R = \frac{1}{C}$.

By Tropp's theorem, we get $Pr(\lambda_{max}(S) \geq 1+\varepsilon) \leq n e^{-\varepsilon^2 C/3} = n e^{-2\log n} = \frac{1}{n}$, as $C = 6\log n / \varepsilon^2$.

The lower tail follows similarly.

So, with probability at least $1 - \frac{2}{n}$, we have $\lambda_{max}(S) \geq 1+\varepsilon$ and $\lambda_{min}(S) \leq 1-\varepsilon$, and so $(1-\varepsilon)I \preceq S \preceq (1+\varepsilon)I$, proving that our solution $S$ is a $(1\pm\varepsilon)$ spectral sparsifier of $I_n$.

By a union bound, we know that a $(1\pm\varepsilon)$-spectral sparsifier with $O(n \log n / \varepsilon^2)$ edges exist, and indeed the random sampling algorithm will succeed with high probability, proving the theorem.

## Discussions

There are a few things to discuss about.

① By considering this linear algebraic generalization of cut sparsification, we have a clean and arguably simple proof of the result of Benczur and Karger.

A subsequent amazing result by Batson, Spielman and Srivastava proves that every graph has a $(1\pm\varepsilon)$-spectral sparsifier with $O(n/\varepsilon^2)$ edges, which is best possible.

We don't know of an alternative (combinatorial) proof to achieve the same bound even for cut approximator (a special case).

This linear algebraic perspective seems to be the correct way to look at the problem.

② The sampling probability $p_e$ is directly proportional to the effective resistance of the edge $e$.

Recall that $p_e = \|v_e\|_2^2 = \|L_G^{+} b_e\|_2^2 = b_e^T L_G^{+} b_e$. Let $e=uv$.

Note that $L_G^{+} b_e$ is a solution $x$ to $L_G x = b$, which is the potential vector $\vec{\phi}$ of the electrical flow problem when one unit of electrical flow is sent from $u$ to $v$.

Then, $b_e^T L_G^{+} b_e = b_e^T \vec{\phi} = \phi(u) - \phi(v)$ is just the definition of $R_{eff}(u,v)$.

So, the sampling algorithm works by sampling each edge with probability proportional to its effective resistance, a somewhat surprising application of this concept (though it makes sense).

③ There is a nearly linear time algorithm to estimate the effective resistances of all edges.

The main tools are a near linear time algorithm to solve a Laplacian system of equations (another breakthrough result by Spielman and Teng), and also dimension reduction.

So, we have a near linear time (randomized) algorithm for constructing spectral sparsifiers.

These results are also within the scope of this lecture, but we don't have time.

④ The analysis of the random sampling algorithm is tight.

In a complete graph, the effective resistance of every edge is the same, as the graph is symmetric.

So, the random sampling algorithm on a complete graph is just the uniform sampling algorithm.

We know from homework 1 (although informally) that it won't work with $o(n\log n/\varepsilon^2)$ edges.

---

## Near linear time algorithm for Laplacian solver

This one will be very brief.

Spielman and Teng were the first one to design a near linear time algorithm to solve $Lx=b$.

The original algorithm is very complex.

Since then, many algorithms have been discovered.

A simple one is to solve the electrical flow problem by keep "fixing" cycles. (See L27 for CS270.)

A recent algorithm is very simple and elegant, which is a variant of the Gaussian elimination method.

When we do Gaussian elimination, we could make the matrix dense (called fill-in).

For a Laplacian matrix, we know exactly what a Gaussian elimination would do to the graph.

This will remove a vertex and adds a complete graph on its neighbors. So, even we start with a sparse graph, the graph will become dense after a few steps.

The new idea is to sparsify the complete graph added to keep the graph sparse, and hope that a good spectral approximation will give us a good approximate solution.

This idea is promising, but it requires some specific sparsification method and a delicate analysis to prove that it works.

---

## References


- Christiano, Kelner, Madry, Spielman, Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs, 2010.
- Spielman and Srivastava, Graph sparsification by effective resistance, 2008.
- Batson, Spielman, and Srivastava, Twice-Ramanujan Sparsifiers, 2009.
- Kelner, Orecchia, Sidford, Zhu. A simple combinatorial algorithm for solving SDD systems in near linear time.
- Kyng, Sachdeva. Approximate Gaussian elimination for Laplacians: Fast, Sparse, and Simple.