CS 798 - Convexity and Optimization, Winter 2017, Waterloo

Lecture 8: Minimum cut

One goal of the course is to see recent developments in algorithm design using convexity.

Today we will see the first example, using gradient descent (along with Laplacian solver) to design new algorithms for the minimum s-t cut problem, a classical problem in combinatorial optimization.

The material in this lecture is extracted from a paper by Lee, Rao, Srivastava [LRS] from 2013.

Projection

We will start with some background knowledge before presenting the new algorithm.

First, let us start with projection into a subspace $S \subseteq \mathbb{R}^n$.

By projection, we would like to construct an operator $P \in \mathbb{R}^{n \times n}$ such that if $v \in S$, then $Pv = v$, and if $v \notin S$, then $Pv \in S$.

Note that such an operator will satisfy $P^2 = P$.

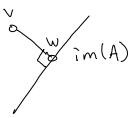
There are many ways to define such an operator, but perhaps the most natural way is to define Pv as the closest point of v in S, i.e. $Pv = \underset{x \in S}{\operatorname{argmin}} \|x - v\|_2$.

This is also the operation that we needed in projected gradient descent, when the feasible set is a subspace. We will get back to this soon.

How do we construct the operator P such that $Pv = \underset{x \in S}{\operatorname{argmin}} \|x - v\|_2$?

Suppose the subspace S is of dimension k and is represented as the image of an $n \times k$ matrix A, i.e. $S = \mathrm{Im}(A)$, where the columns of A form a (not necessarily orthogonal) basis of S.

Given v, for $w \in \mathrm{Im}(A)$ to minimize $\|v - w\|_2$, we must have $(v-w) \perp \mathrm{im}(A)$, as otherwise w is not the closest point to v (recall the proof in the separation theorem in L03).



This is equivalent to $(v-w)^T A = 0 \iff A^T v = A^T w$.

Since $w \in \mathrm{Im}(A)$, it can be written as $w = Ax$ for some x, and the above is equivalent to $A^T v = A^T A x$, and thus $x = (A^T A)^{-1} A^T v$, and hence $w = Ax = A(A^T A)^{-1} A^T v$.

So $P = A(A^T A)^{-1} A^T$ is the projection matrix.

Note that when the columns of A are orthonormal, then $A^T A = I$, and so P is simply AA^T in this case.

Pseudo-inverse

When the columns of A are not linearly independent, the matrix A^TA is not of full rank and its inverse does not exist, as there are many vectors with the same image.

For any matrix M, recall that $\ker(M) := \{x \mid Mx = 0\}$ is the kernel (or nullspace) of M.

Let Q be a real symmetric matrix, with eigen-decomposition $Q = \sum_{i=1}^{n} \lambda_i v_i v_i^T$ (some λ_i could be zero).

The pseudo-inverse $Q^\dagger$ of Q is defined as $Q^\dagger := \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^T$.

Note that $Q^\dagger Q v = v$ for any $v \perp \ker(Q)$, i.e. restricting to the subspace perpendicular to $\ker(Q)$, the mapping of Q is one-to-one and we can define the inverse operation.

Coming back to the projector matrix, when A^TA is not of full rank, then $P = A(A^TA)^\dagger A^T$ is the projector, as $\mathrm{Im}(A^T) \perp \ker(A^TA) = \ker(A)$; please check.

While the pseudoinverse is a useful concept, for the purpose of this course (or at least today), we can just pretend that it is the inverse, without sacrificing our understanding much.

Gradient descent with equality constraints

Consider the optimization problem $\min f(x)$

s.t. $Ax = b$ where $A \in \mathbb{R}^{m \times n}$ for $m \leq n$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$.

In principle, we can always reduce it to an unconstrained optimization problem by a change of variables.

The set $\{x \mid Ax = b\} = x_0 + \ker(A)$ where x_0 is any feasible solution to $Ax = b$.

Let M be a matrix whose columns form a basis of $\ker(A)$, say $M \in \mathbb{R}^{n \times k}$.

Then the set $\{x \mid Ax = b\} = \{x_0 + Mz \mid z \in \mathbb{R}^k\}$.

So, the problem $\min f(x)$ s.t. $Ax = b$ is equivalent to $\min f(x_0 + Mz)$ s.t. $z \in \mathbb{R}^k$.

Then, we can just do gradient descent to solve the unconstrained problem.

The gradient of the unconstrained problem is $M \nabla f(x_0 + Mz)$.

Projected equations

We will use this reduction in the special case when A is a projection matrix P.

Then, obviously we should have $Pb = b$, as otherwise there will be no solution to the system.

Check also that $\ker(P) = \mathrm{span}\{I - P\}$ as $P(I-P) = P - P^2 = 0$ by the definition of projection matrix.

So, $\min f(x)$ s.t. $Px = b$ is equivalent to $\min f(b + (I-P)x)$ s.t. $x \in \mathbb{R}^n$.

At a feasible point x such that $Px=b$, the gradient in the unconstrained problem is simply $(I-P)\nabla f(b+(I-P)x) = (I-P)\nabla f(x)$.

The smoothness constant β becomes the minimum β such that $\|(I-P)\nabla f(x) - (I-P)\nabla f(y)\| \le \beta \|x-y\|$.

By setting $x' = x - \eta(I-P)\nabla f(x)$, we still have x' being feasible.

So, the algorithm can be implemented directly in the original constrained setting.

- start with a feasible solution x_0, i.e. $Px_0 = b$
- repeat $x_{t+1} = x_t - \frac{1}{\beta}(I-P)\nabla f(x_t)$

On the other hand, since the algorithm corresponds to the gradient descent algorithm in the unconstrained setting, the analysis in the previous lecture still holds, e.g. $O(\frac{\beta}{\varepsilon})$ iterations.

Accelerated gradient descent

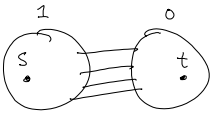
For the min-cut algorithm to be competitive, we use Nesterov's accelerated algorithm.

Theorem (Nesterov) Let $S = \{x \mid Px=b\}$ and f be a convex function. Given starting point $x_0 \in S$, and gradient oracle $(I-P)\nabla f(x)$, the accelerated algorithm produces a solution x_T with $f(x_T) - f(x^*) \le \varepsilon$ in $T = O\left(\sqrt{\frac{\beta}{\varepsilon}}\, \|x_0 - x^*\|_2\right)$ iterations where β is smoothness of $(I-P)\nabla f(y)$.

Linear program

The minimum s-t cut problem can be formulated as a particularly simple linear program.

Recall that the min s-t cut problem is as follows: given an undirected graph $G=(V,E)$ and two specified vertices $s,t \in V$, find a minimum set of edges $F \subseteq E$ such that s and t are disconnected when we remove F from G.



For each vertex $v \in V$, we have an indicator variable $x_v \in \{0,1\}$.

The following integer linear program is an exact formulation of the min s-t cut problem.

$$\min \sum_{uv \in E} |x_u - x_v|$$

$$\text{s.t. } x_s = 1,\ x_t = 0,\ \text{and } x_v \in \{0,1\} \text{ for } v \in V.$$

Every integral solution defines a cut by putting vertices with value 1 on one side and vertices with 0 on the other side. Then the objective is counting the crossing edges.

An integer linear program is not convex, and is NP-hard to solve in general.

We can write the following linear programming relaxation of the problem:

$$\min \sum_{uv \in E} |x_u - x_v|$$

$$\text{s.t.} \quad x_s - x_t = 1$$

will show in class ↓

Okay, it is not a linear program in this form, but we can rewrite it as a linear program (exercise). This is essentially the same program but with the integral constraints removed. Perhaps surprisingly, there is still always an integral optimal solution to this linear program.

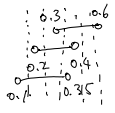
Claim If the objective value is p, then we can find an s-t cut with at most $\lfloor p \rfloor$ edges in $O(|E|)$ time.

proof For $0 \le l \le 1$, we consider s-t cuts of the form $S_l := \{ v \mid x_v \ge l \}$, the "level sets".

We claim that there is at least one l such that the cut $(S_l, V - S_l)$ has at most $\lfloor p \rfloor$ edges.

To see this, note that by a "row-column" argument, the objective value can be rewritten as

$\sum_{uv \in E} |x_u - x_v| \ge \int_0^1 |\delta(S_l)| \, dl$, where $\delta(S_l)$ denotes the set of edges "crossing" S_l, as each edge is still counted at least $|x_u - x_v|$ in the RHS.



In the picture, I am trying to show that the LHS is counting by summing the length of the "rows", and the RHS is counting by summing the number of edges in each "column".

So, if by contradiction that $|\delta(S_l)| > \lfloor p \rfloor$ for all l, then the RHS will be greater than p, contradicting the assumption that the LHS is equal to p.

It is an exercise to check that searching for all S_l can be done in $O(|E|)$ time.

(People often just pick a random l and argue that the expected value is p.) □

With the claim, we know that we can solve min s-t cut by solving this linear program, but it is not something that gradient descent is good at.

Gradient descent

Recall that the problem now is to solve $\min \sum_{uv \in E} |x_u - x_v|$ s.t. $x_s - x_t = 1$.

It looks simple, but when you write it as a "real" LP, we need to introduce extra variables and some inequality constraints on each edge, and this is not easily handled in the gradient descent framework.

We may also try to do gradient descent directly, but the objective function is not differentiable.

A cool idea of [LRS] is to approximate the objective function by a smooth function.

Consider the following reformulation $\min \sum_{uv \in E} \sqrt{(x_u - x_v)^2 + \mu^2}$, where μ is a parameter.

$\text{s.t.} \quad x_s - x_t = 1$

(think of μ being a small number)

There are two advantages of using the new objective functions:

① It is now differentiable and the smoothness parameter can be computed.

② The approximation error can be easily analyzed.

As we will see, this is not our final formulation, but let me elaborate on the second point first.

Error analysis

Note that $|x_u - x_v| \le \sqrt{(x_u - x_v)^2 + \mu^2} \le |x_u - x_v| + \mu$.

Suppose a minimum cut has F edges. Then, this corresponds to a solution with objective value F in the linear program, and by the above (second) inequality the same solution has objective value at most $F + \mu|E|$ using the new objective function.

Suppose we use gradient descent to solve the new problem up to an additive δ error.

Then we can find a solution with objective value at most $F + \mu|E| + \delta$, and thus by the above first inequality this gives us a solution x with $\sum_{uv \in E} |x_u - x_v| \le F + \mu|E| + \delta$, and we know from a previous claim that we can extract from x an s-t cut with $\le F + \mu|E| + \delta$ edges.

So, suppose we would like to find an s-t cut with at most $(1+\varepsilon)F$ edges, i.e. an $(1+\varepsilon)$-approximation.

We could set $\mu = \frac{\varepsilon F}{2|E|}$ and $\delta = \frac{\varepsilon F}{2}$ so that $F + \mu|E| + \delta = (1+\varepsilon)F$.

Running time estimate

Let's do a quick and dirty run-time analysis.

To achieve additive δ error, Nesterov's algorithm requires $O\left(\sqrt{\frac{\beta}{\delta}}\, \|x_0 - x^*\|_2\right)$ iterations.

The smoothness parameter will depend on μ. Let's say $\beta = \frac{1}{\mu}$. We will justify it later.

Plugging in $\mu = \frac{\varepsilon F}{2|E|}$ and $\delta = \frac{\varepsilon F}{2}$, we need $O\left(\frac{\sqrt{|E|}}{\varepsilon F}\, \|x_0 - x^*\|_2\right)$ iterations.

I don't have a good bound on $\|x_0 - x^*\|_2$, so just use a crude bound $\|x_0 - x^*\|_2 \le \sqrt{|V|}$.

Then, the number of iterations is thus $O\left(\sqrt{|V||E|} \,/\, (\varepsilon F)\right)$.

Each iteration requires a gradient computation and needs $O(|E|)$ time, so total running time is $O\left(\frac{|E|^{1.5}\sqrt{|V|}}{\varepsilon F}\right)$.

We get something, but not very competitive, as we know combinatorial algorithms solving the problem exactly in $O(|E|^{1.5})$ time.

Here, even if $F = |V|$, the running time is still $O\left(\frac{|E|^{1.5}}{\varepsilon}\right)$.

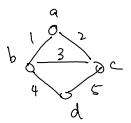
Notice, however, the gradient descent algorithm has the interesting property that it runs faster when the optimal value F is bigger. This is definitely not true for augmenting path algorithms.

Changing space

The crucial idea in [LRS] is to do a change of the variables, so that we optimize over the edge space in $\mathbb{R}^{|E|}$ instead of over the vertex space $\mathbb{R}^{|V|}$.

Basically, it means that we will use a new variable y_e to replace $x_i - x_j$ for $e=ij$.

Let B be the $|E| \times |V|$ matrix such that the i-th row corresponds to the i-th edge. Let the i-th edge be $e=uv$. Then the i-th row of B is $x_u - x_v$ where x_u is the vector in $\mathbb{R}^{|V|}$ with one in the u-th position and zero otherwise.



$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & -1 & & \\ 1 & & -1 & \\ & & 1 & -1 \\ & 1 & & -1 \\ & & 1 & -1 \end{bmatrix} \end{array}$$

Then, the objective $\sum_{uv \in E} |x_u - x_v|$ can be rewritten as $\|Bx\|_1$.

If we use $y \in \mathbb{R}^{|E|}$ to denote Bx, then the problem $\min \sum_{uv \in E} |x_u - x_v|$ s.t. $x_s - x_t = 1$ can be rewritten as $\min \sum_{e \in E} |y_e|$ s.t. $y = Bx$ for some x such that $x_s - x_t = 1$.

Eventually, we want to rewrite the constraint ($y = Bx$ for some x such that $x_s - x_t = 1$) as a system of linear equalities $Py = b$ where P is a projection matrix, so that we can use the gradient descent algorithm with equality constraints discussed above.

To do this, we need a little detour to introduce Laplacian matrices.

Laplacian matrix

The $|V| \times |V|$ matrix L defined by $B^T B$ is called the Laplacian matrix of the graph.

It can be written as $L = \sum_{uv \in E} (x_u - x_v)(x_u - x_v)^T$, or more explicity as $D - A$, where D is the diagonal degree matrix (i.e. $D_{v,v} = \deg(v)$) and A is the adjacency matrix of the graph (i.e. $A_{u,v} = 1$ if $uv \in E$).

The all-1 vector is in the kernel of A.

When the graph is connected, the kernel is just the one-dimensional space spanned by the all-1 vector.

So, the pseudo-inverse $L^\dagger$ can invert any vector that is perpendicular to the all-one vector.

A very important recent result is that $L^\dagger b$ can be solved in near linear time $\tilde{O}(|E|)$, where the $\tilde{O}$ notation hides some polylog factor.

Theorem (Laplacian solver) (informal) $Lx = b$ can be solved in $\tilde{O}(|E|)$ time.

This is informal because the running time depends on the error, but we just pretend that it is exact!

There is a long and interesting story of this result. We may discuss more during class.

It is equivalent to computing electrical flow in a resistor network, but we don't need this interpretation today.

See L13 of CS798 in 2015 for one proof, and the course project page for more references.

Dan Spielman will come to Waterloo in late March to give a distinguished lecture about Laplacian solver!

Continuing changing space

Recall that we want to rewrite the constraint ($y = Bx$ for some x such that $x_s - x_t = 1$) as a system of linear equalities $Py = b$ where P is a projection matrix.

The constraint $y = Bx$ for some x is saying that $y \in \mathrm{Im}(B)$, and we can use a projector for it.

Recall that $\Pi := B(B^TB)^\dagger B^T$ is a projector into $\mathrm{Im}(B)$.

So, $y \in \mathrm{Im}(B)$ can be expressed as $\Pi y = y$, or equivalently $(I - \Pi)y = 0$.

We have the additional requirement that $y = Bx$ for some x such that $\langle x, \chi_s - \chi_t \rangle = 1$.

If B is an invertible matrix, we can express it as $\langle Bx, B^{-1}(\chi_s - \chi_t)\rangle = 1 \Leftrightarrow \langle y, B^{-1}(\chi_s - \chi_t)\rangle = 1$.

But B is not even a square matrix, so we need to do something a little more complicated:

$$\langle x, \chi_s - \chi_t \rangle = 1 \Leftrightarrow \langle x, B^TB(B^TB)^\dagger(\chi_s - \chi_t)\rangle = 1 \quad \text{since } \chi_s - \chi_t \perp 1 \text{ and so can be inverted by } L^\dagger$$

$$\Leftrightarrow \langle x, B^TBL^\dagger(\chi_s - \chi_t)\rangle = 1$$

$$\Leftrightarrow \langle Bx, BL^\dagger(\chi_s - \chi_t)\rangle = 1$$

$$\Leftrightarrow \langle y, y_{st}\rangle = 1 \quad \text{where we denote } y_{st} = BL^\dagger(\chi_s - \chi_t).$$

So, the constraint ($y = Bx$ for some x such that $x_s - x_t = 1$) can now be rewritten as

$(I - \Pi)y = 0$ and $\langle y, y_{st}\rangle = 1$, totally eliminated x from the constraint.

Check that $\Pi y_{st} = y_{st}$ and so $(I - \Pi)y_{st} = 0$, i.e. y_{st} is orthogonal to the subspace of $I - \Pi$.

Hence, the two constraints can be combined as $Py = b$ where $P = (I - \Pi) + \frac{y_{st}y_{st}^T}{\|y_{st}\|^2}$ and $b = \frac{y_{st}}{\|y_{st}\|^2}$.

Any solution satisfying $Py = b$ must be of the form $\frac{y_{st}}{\|y_{st}\|^2} + z$ where $(I - \Pi)z = 0$ and $y_{st} \perp z$.

Therefore, $\frac{y_{st}}{\|y_{st}\|^2}$ is the solution to $Py = b$ with the minimum 2-norm. This is a crucial fact later.

So, we have successfully written everything in terms of y.

Final formulation

The original program is $\min \sum_{uv \in E} |x_u - x_v|$

$$\text{s.t.} \quad x_s - x_t = 1.$$

After changing space, with all the above derivation, the program becomes:

$$\min \sum_{e \in E} |y_e|$$

$$\text{s.t.} \quad Py = b \qquad \text{where} \quad P = (I - \Pi) + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2} \quad \text{and} \quad b = \frac{y_{st}}{\|y_{st}\|^2}.$$

This is of the form where we know how to do gradient descent with projector equality constraints.

We add the trick of smoothing the objective function and obtain our final program:

$$\min \sum_{e \in E} \sqrt{y_e^2 + \mu^2}$$

$$\text{s.t.} \quad Py = b.$$

Algorithm and analysis

We just do the gradient descent algorithm with projection equality constraint.

- Initial point $\frac{y_{st}}{\|y_{st}\|^2}$. (check that it is feasible)
- Do gradient descent with direction $(I - P) \nabla g(y)$ where $g(y) = \sum_{e \in E} \sqrt{y_e^2 + \mu^2}$ is the objective function.

Note that $I - P = I - \left(I - \Pi + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2}\right) = \Pi + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2} = B(B^T B)^\dagger B^T + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2} = B L^\dagger B^T + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2}$.

Gradient, Hessian, Smoothness

The gradient $\nabla g(y)$ is easy to compute:

$$\frac{\partial}{\partial y_e} g(y) = \frac{\partial}{\partial y_e} \sum_{e \in E} \sqrt{y_e^2 + \mu^2} = \frac{\partial}{\partial y_e} \sqrt{y_e^2 + \mu^2} = \frac{y_e}{\sqrt{y_e^2 + \mu^2}}.$$

$$\frac{\partial^2}{\partial y_e} g(y) = \frac{\partial}{\partial y_e} \left(\frac{y_e}{\sqrt{y_e^2 + \mu^2}}\right) = \frac{-y_e^2}{(y_e^2 + \mu^2)^{\frac{3}{2}}} + \frac{1}{(y_e^2 + \mu^2)^{\frac{1}{2}}} = \frac{\mu^2}{(y_e^2 + \mu^2)^{\frac{3}{2}}}.$$

Note that the Hessian matrix is just a diagonal matrix with entries $\frac{\mu^2}{(y_e^2 + \mu^2)^{\frac{3}{2}}} \le \frac{\mu^2}{(\mu^2)^{\frac{3}{2}}} = \frac{1}{\mu}$.

So, the maximum eigenvalue of the Hessian is at most $\frac{1}{\mu}$, and so the smoothness parameter is $\frac{1}{\mu}$.

Running time of each iteration

Once we have the gradient $\nabla g(y)$, $(I - P) \nabla g(y) = \left((B L^\dagger B^T) + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2}\right) \nabla g(y)$ can be computed in near linear time.

To see this, $B^T \nabla g(y)$ can be computed in $O(E)$ time, since B has only $2|E|$ nonzero entries.

Once $B^T \nabla g(y)$ is computed, $L^\dagger (B^T \nabla g(y))$ can be computed in near linear time, by solving $Lx = B^T \nabla g(y)$ and the solution $x = L^\dagger (B^T \nabla g(y))$. Finally, Bx can be done in $O(E)$ time as B is sparse.

So, each iteration can be implemented in $\tilde{O}(|E|)$ time. This is an important part about the efficiency.

Analysis

Using Nesterov's accelerated gradient descent, it requires $O\left(\sqrt{\frac{\beta}{\delta}}\,\|y_0 - y^*\|_2\right)$ iterations to achieve δ error.

As before, we set $\mu = \frac{\varepsilon F}{2|E|}$ and $\delta = \frac{\varepsilon F}{2}$ so that we get the objective value $\le F + \mu|E| + \delta = (1+\varepsilon)F$.

Since $y = Bx$, we can get back $x = L^+ B^T y$, and then obtain from it an s-t cut with $\le (1+\varepsilon)F$ edges.

The smoothness parameter $\beta \le \frac{1}{\mu}$ as we just calculated.

Everything looks the same as before we changed space, now comes the reason of changing space.

Recall that $\frac{y_{st}}{\|y_{st}\|^2}$ is the solution to $Py = b$ with the minimum norm.

It is the projection of the origin to the solution space.

So, $\|y_0 - y^*\|_2 = \|\text{proj}(0) - \text{proj}(y^*)\|_2$ where $\text{proj}(y') := \underset{y}{\text{argmin}}\{\|y - y'\|_2 \mid Py = b\}$ is the projection to solution space.

$\le \|0 - y^*\|_2$ since projection won't increase distance (check this)

$= \sqrt{F}$ since optimal solutions have F edges.

This is a drastic improvement over the previous bound before we changed space.

So, the number of iterations required is $O\left(\sqrt{\frac{\beta}{\delta}}\,\|y_0 - y^*\|_2\right) = O\left(\frac{1}{\varepsilon}\sqrt{\frac{|E|}{F}}\right)$.

The total running time is $\tilde{O}\left(\frac{1}{\varepsilon}\frac{|E|^{1.5}}{\sqrt{F}}\right)$.

To summarize, the whole point of changing space is to get an improved initial distance bound, and the amazing Laplacian solver ensures that each (more complicated) step can still be done in $\tilde{O}(|E|)$.

The reason that we would like to change space is that the solution in the y-space has a small support (F nonzeros), while we have no control over the x-space (e.g. the min-cut could be a bisection with $\frac{|V|}{2}$ nonzeros)

Improvements

Recall that this has the surprising property that it is faster when F is bigger.

There is a combinatorial algorithm by Karger and Levine with running time $\tilde{O}(|E| + |V|F)$.

Balancing the two algorithms by choosing $F = \frac{m}{(n\varepsilon)^{\frac{2}{3}}}$ (e.g. use combinatorial when $F \le \frac{m}{(n\varepsilon)^{\frac{2}{3}}}$ and descent otherwise), the runtime of the best of the two is $O(mn^{\frac{1}{3}}\varepsilon^{-\frac{2}{3}})$, which was the best known in 2013.

More recently, the continuous approach (combined with combinatorial subroutines) has pushed the running time to $\tilde{O}(|E|)$, using a projected gradient descent approach.

There are some (current) shortcomings of the continuous approach, e.g. not applicable in directed graphs and not efficient for weighted graphs (while combinatorial approaches could do).

Recently there are some progress, but there are still many open questions in this area.

Comments

Unlike the classical results, the new results will probably be not as important (eg. this result is already subsumed).

The new results are perhaps not as easy to follow, as they are not well understood yet.

And you may wonder why you need to care about approximating min s-t cut in unweighted undirected graphs!

These are all reasonable comments, but they are off the point.

Talking about these new results in details is just a way to illustrate some nice technical ideas through some nice stories.

Through this min-cut example, we take the opportunity to talk about some useful background knowledge such as projection, gradient descent with equality constraints, and integral LP.

Of course, we also learn the cool idea of smoothing the objective function, the Laplacian solver, and some low level details about changing variables.

A very important message that I want to come across is that relatively simple methods from convex optimization can get improvements for well-studied problems in the combinatorial world, and I think now it is the right time to learn more about these ideas.

These are the useful bits that are important, not the theorems themselves, but of course I will try to cover interesting theorems as well.

Reference: [LRS] A new approach to computing maximum flow using electrical flow. by Lee, Rao, Srivastava.