Today we will see an interesting application of the Laplacian solver and electric flow to the maximum flow problem, a classical problem in combinatorial optimization.

Then we quickly summarize this course.

**Solving Linear Program by Multiplicative Weights Update Method** \([1, 2]\)

We will formulate the maximum flow problem as a linear program, and apply the multiplicative update method to approximately solve it quickly.

So first let us discuss how to solve linear programs by the multiplicative update method.

The usual form of linear programs is:

\[
\begin{align*}
\text{max } & \quad C^T x \\
\text{subject to } & \quad A x \leq b
\end{align*}
\]

In our setting we only consider the problem of finding a solution \(x\) that satisfies \(Ax \leq b\), or determine that no solutions exist. This is without loss of generality as we can do binary search on the objective value and then rewrite the objective function as a constraint. If we know that the range of the objective value is in \([-M, M]\), then the optimization problem can be reduced to \(O(\log M)\) decision problems, to get a good approximation of the objective value. It is not too slow if \(M\) is not too big, which will be the case in the maximum flow problem.

So our task is to find an \(x\) such that \(Ax \leq b\) and \(x \in P\), where \(P\) is a convex set.

Usually we put the constraints that are "easy" to satisfy to \(P\), and focus on the "hard" constraints in \(Ax \leq b\).

To apply the multiplicative update method, we think of this as a "game" between us and the constraints, where our objective is to find a solution that minimizes the violation of the constraints, and each constraint is a player and their objective is to maximize the violation.

It is difficult for us to deal with so many players at the same time, and here is where the multiplicative update method comes in.

As we saw last week, there is a "multiplicative update player", who plays a mixed strategy of the constraint players, and his performance is almost as good as the best constraint player in hindsight.

So, if we can always find a strategy to beat the multiplicative update player, then we know that
we know that our strategies also do very well against the best constraint player.

In our setting, the multiplicative update player plays a mixed strategy of the constraints, which is just a linear combination of the constraints. This is just one constraint instead of many constraints.

So, if we can always find a strategy to beat the multiplicative update player, meaning that if we can always find a solution to satisfy that one constraint, then we know that the average solution will be an almost feasible solution to the linear program.

---

**Packing Program** [1,3]

We illustrate how to solve linear programs by the above framework in more details.

For simplicity we consider “packing programs” $Ax\leq b$ and $x \geq 0$, where each entry of $A$ and $b$ is non-negative.

This will be enough for our purpose of approximating maximum flow.

We can put $x \geq 0$ into $P$, the set of easy constraints, and so we write the problem as $Ax\leq b$, $x \in P$.

Note that the objective function, say $\sum \frac{x_i}{a_i}$, is also in $P$, otherwise the linear program is trivially feasible.

We model this problem as a game as above.

Each constraint $Ax \leq b_i$ is an expert. Given an $x$, the payoff to constraint $i$ is $Ax_i/b_i$.

Our objective is to play an $x$ that minimizes payoff, while the constraints’ objective is to maximize payoff.

This is a two player zero sum game that we studied last time.

We can find a good solution by the following algorithm.

**Algorithm**

Initially the weight for each constraint is 1, i.e. $w_i = 1$ for each constraint $i$.

Repeat for $T = \frac{\ln n}{C^2}$, where $P$ is a bound on the maximum payoff, and $n$ is number of constraints.

- The multiplicative update player plays the constraint $\frac{1}{T} \sum \frac{x_i}{a_i} \leq \frac{1}{T} \sum \frac{x_i}{b_i}$.

- We determine whether there exists $x \in P$ that satisfies the constraint. If yes we respond with $x$.

  If no then we say “no solution for $Ax\leq b$ and $x \in P$”.

- The multiplicative player updates the weights of the constraints according to the payoff $Ax_i/b_i$ for each $i$.

We output $x = \frac{1}{T} \sum x_i$ as our solution.

**Analysis**

Think on average about the correctness of the algorithm.

L14 Page 2
First we argue about the correctness of the algorithm when it says no.

If there is a solution to \( Ax \leq b \), then the same solution \( x \) must satisfy \( \overline{\nu}(w; A_1) x \leq \overline{\nu}(w; b_1) \).

So, if there is no solution to \( \overline{\nu}(w; A_1) x \leq \overline{\nu}(w; b_1) \), then there must be no solution to \( Ax \leq b \).

So, the algorithm says no correctly, hence we just need to show that if the algorithm didn't say no, then the returned solution is an almost feasible solution.

As we always respond with \( x_t \) that satisfies the constraint given by the multiplicative update player, the total gain of the multiplicative update player is at most \( T \).

By the analysis of the multiplicative update method last week, this implies that the total gain of the best constraint is at most \( \frac{T + p_{\text{max}}}{1 - \varepsilon} \), where \( p_{\text{max}} \) is the maximum payoff during the game.

This means that \( \frac{\sum_i A_i x_t}{b_i} \leq \frac{T + p_{\text{max}}}{1 - \varepsilon} \), thus \( (1 - \varepsilon) \cdot \frac{\sum_i A_i x_t}{b_i} \leq 1 + \varepsilon \) as \( T = \frac{p_{\text{max}}}{\varepsilon} \).

Hence \( A_i x_t \leq (\frac{1 + \varepsilon}{1 - \varepsilon}) \cdot b_i \leq (1 + 4\varepsilon) b_i \) for all \( i \) for \( \varepsilon \leq \frac{1}{2} \).

**Complexity**

Thus, if \( Ax \leq b \) and \( x \in P \) is feasible, the above algorithm finds an almost feasible solution in \( O\left( \frac{p_{\text{max}}}{\varepsilon} \right) \) iterations.

For this algorithm to be efficient, we need to implement an "oracle" with the following properties:

1. Given a constraint \( C: x \leq d \), determine whether there exists \( x \in \mathbb{P} \) that (almost) satisfies it.
2. Always return \( x_t \) such that \( \frac{\sum_i A_i x_t}{b_i} \leq p_{\text{max}} \), which should be as small as possible.
3. The oracle can be implemented efficiently.

The parameter \( p_{\text{max}} \) is called the "width" of the oracle.

---

**Maximum Flow in Undirected Graphs** \([3, 4]\)

Now we specialize the above algorithm to the maximum flow problem.

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

\[
\max \sum_{e \in E^+} f_e \quad \text{ (where } E^+ \text{ is the set of directed edges going out from } s) \\
\sum_{e \in \delta^+(v)} f_e = \sum_{e \in \delta^-(v)} f_e \quad \text{ for all } v \in V - \{s, t\} \quad \text{ (where } \delta^+(v) \text{ is the set of incoming edges to } v) 
\]
\[
\sum_{e \in \mathcal{E}(v)} f_e = \sum_{e \in \mathcal{E}_v} f_e \text{ for all } v \in V - \{s,t\} \quad \text{(where } \mathcal{E}_v \text{ is the set of incoming edges to } v) \\
f_e \leq c_e \text{ for all } e \\
f_e \geq 0 \text{ for all } e
\]

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 \( \sum_{e \in \mathcal{E}(v)} f_e = k \).

To apply the above method of approximately solving linear programs to approximately solving the maximum flow problem, we first identify the easy constraints. There is a tradeoff here: if you put more constraints in \( P \), then it is easier to solve the remaining linear program but harder to design the “oracle.”

It turns out that electric flow will provide us a powerful oracle that can take care of most constraints.

We set \( P = \{ \text{objective constraint, flow conservation constraints, } \text{ non-negative constraints}\} \), and just leave the capacity constraint out.

Given this \( P \), the maximum flow problem is reduced to \( O(\log m) \) subproblems of the form:

\[
f_e \leq 1 \text{ for all edge } e \\
\sum_{e \in P} f_e = k
\]

The only constraints left are the capacity constraints. This fits into above algorithm for packing programs.

So, by the above analysis, we can find an almost feasible solution (i.e. the capacity is at most \( 1+\epsilon \)) in \( O\left(\frac{\log n}{\epsilon^2}\right) \) calls to the oracle with the following properties:

1. \( \sum_{e \in P} f_e \leq (1+\epsilon) \sum_{e \in P} w_e \). This is the first requirement of the oracle in the above section.

The multiplicative update player will always give a constraint of the form \( \sum_{e \in \mathcal{E}_v} w_{e} f_e \leq \frac{n}{2} w_{e} \).

In the above analysis we assume that we can always respond with a solution that satisfies the constraint, but it is straightforward to check that it is enough to respond with an almost feasible solution, i.e. \( \sum_{e \in P} f_e \leq (1+\epsilon) \sum_{e \in P} w_e \), since we are doing approximations anyway.

2. \( f \in P \). This is the second requirement of the oracle in the above section.

This is the maximum payoff. If it is large then the convergence rate is slower.

It is clear that in our problem we want to bound it: if \( f_e = M \) for a large \( M \), then it takes
at least $M$ iterations for the average value to be at most $1$, the capacity of the edges.

Oracle by Electric Flow \cite{3,4}

We show that the electric flow algorithm can be used to obtain an oracle with the following properties:

1. The oracle returns $f$ that satisfies flow conservation constraints, non-negative constraints, and objective value constraint.
2. When the maximum flow problem is feasible, the oracle can always return $f$ satisfying $\sum_{e \in E} f_e \leq (1+\varepsilon) \sum_{e \in E} c_e$.
3. When the maximum flow problem is feasible, the oracle can always return $f$ with $f_e \leq O(\sqrt{m})$.
4. The oracle can be implemented in $O(m)$ time.

Assuming such an oracle exists, then we can get a flow of optimal value in $O\left(\frac{\ln n}{\varepsilon^2}\right) = O(\sqrt{m})$ iterations, where the capacity of each edge is violated by a factor of $1+\varepsilon$. By scaling down the flow values by a factor of $1+\varepsilon$, we obtain a flow of value $\frac{\text{OPT}}{(1+\varepsilon)}$ where all the edge capacities are satisfied.

The total running time is $O(\sqrt{m}) \cdot O(m) = O(m^{1.5})$ as there are $O(\sqrt{m})$ iterations and each iteration takes only $O(m)$ time.

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 $R_e = \min \left\{ e + \frac{c_e}{1+\varepsilon}, \frac{W}{\varepsilon} \right\}$ where $W = \sum_{e \in E} c_e$, 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, we is updated by $W_e = W_e (1+\frac{\varepsilon}{\sqrt{m}} f_e)$ 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.

1. It is clear by construction that the flow conservation constraints, the non-negativity constraints, and the objective value constraint are all satisfied.
2. This is the most interesting part.

   If the maximum flow problem is feasible, then there is a flow of $k$ units from $s$ to $t$, without violating the capacity constraints.
The total energy of this flow is at most \( \frac{3}{4} \frac{e^3}{m} e \gamma e \leq \frac{3}{4} \frac{e}{m} (\frac{e}{m} \frac{e}{m}) = (1+\varepsilon)W. \)

Since electric flow \( f \) minimizes the energy, we must have \( \frac{3}{4} e^3 \frac{e}{m} e \leq (1+\varepsilon)W \), or otherwise we can conclude that the maximum flow problem is infeasible.

We would like to bound \( \frac{3}{4} e^3 \frac{e}{m} e \), and in this case for so long we would try Cauchy-Schwarz.

\[
\left( \frac{3}{4} e^3 \frac{e}{m} e \right)^2 \leq \left( \frac{3}{4} e e \right) \left( \frac{3}{4} e e \right) \leq \left( \frac{3}{4} e e \right) \left( \frac{3}{4} e e \right) = (1+\varepsilon)^2 W^2,
\]

and so we have

\( \frac{3}{4} e^3 \frac{e}{m} e \leq (1+\varepsilon)^2 W, \) as required.

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

\[
\Rightarrow \frac{e^3 \frac{e}{m} e}{m} \leq (1+\varepsilon)W \Rightarrow \frac{e}{m} e \leq \left( \frac{4+\varepsilon}{3} \right) m = O(1/m), \text{ as required.}
\]

As we saw in week 12, computing electric flow is the same as solving Laplacian systems, which we saw in last week that can be done in \( O(m) \) time. I should point out that it only gives an approximation, and it takes some work to modify the approximate solution to satisfy the flow conservation constraints, but we will just happily ignore those details.

This completes the \( O(m^{10}) \) time algorithm to give an \( (1+\varepsilon) \)-approximation of the maximum flow problem in undirected graphs, matching the best known combinatorial method.

Next we present an improvement to \( \tilde{O}(m^{9/8}) \) time.

**Faster Algorithms \([3,4]\)**

One bottleneck of the algorithm is that it takes \( O(1/m) \) iterations, because the flow of an edge may be as high as \( \omega(1/m) \). Consider the example where there are \( k \) disjoint paths of length \( k \) between \( s \) and \( t \), and there is one edge between \( s \) and \( t \). Then there will be \( (k+1)/2 \) units of flow on the edge between \( s \) and \( t \). As there are \( k+1 \) edges in the graph, the flow on that edge is \( O(1/m) \).

The idea of the improved algorithm is to delete the edges with too much flow going across them.

This actually sounds counterintuitive, as these edges seem to be in optimal solutions, but it turns out that there will not be many of them, and so deleting them will not decrease the optimal value by much, while improving the convergence rate of the multiplicative update algorithm.

**Modified Algorithm**

The algorithm is the same except that whenever there is an edge with electric flow value greater than \( \varepsilon \),
then we delete this edge from the graph.

**Effective Conductance**

To analyze the effect of deleting an edge, we need the concept of effective conductance.

We know by the Thomson's principle that the electric flow is the flow that minimizes energy, among all the flow that satisfies flow conservation constraints.

\[ \mathcal{E}(f) = \sum_{uv} \mathcal{F}_{uv} \mathcal{P}_{uv} = \sum_{uv} \frac{(\phi_u - \phi_v)^2}{R_{uv}}, \] where \( \phi_u \) denotes the voltage on \( u \).

If the flow \( f \) is an electric flow of 1 unit from \( s \) to \( t \), then \( \mathcal{R}_{eff}(s,t) = \mathcal{E}(f) \).

Recall that \( \mathcal{R}_{eff}(s,t) = \phi_s - \phi_t \).

By setting \( \phi_t = 0 \) and scaling \( \phi_s = 1 \), we have a flow of \( \sqrt{\mathcal{R}_{eff}(s,t)} \) unit from \( s \) to \( t \), and the energy of this flow is equal to \( \mathcal{E}(f)/\mathcal{R}_{eff}(s,t) = \sqrt{\mathcal{R}_{eff}(s,t)} \), as the electric flow of 1 unit from \( s \) to \( t \) has energy \( \mathcal{E}(f)/c^2 \) by scaling the flow on each edge by a factor of \( c \).

Among all vectors on vertices, the voltage vector is the one that minimizes energy.

This is a "dual" version of Thomson's principle.

**Theorem** Let \( r \) be the vector of resistances. Let \( \mathcal{C}_{eff}(r) = \min_{\phi} \sum_{uv} \frac{(\phi_u - \phi_v)^2}{R_{uv}} \).

Then \( \mathcal{C}_{eff}(r) \) is minimized by the voltage vector of the electric flow of \( \sqrt{\mathcal{R}_{eff}(s,t)} \) units from \( s \) to \( t \).

**Proof** We have seen that the value \( \sqrt{\mathcal{R}_{eff}(s,t)} \) is attainable by that voltage vector.

The partial derivative on a variable \( \phi_v \), \( \frac{\partial \mathcal{E}(f)}{\partial \phi_v} = \sum_{v \in V \setminus \{s,t\}} \frac{(\phi_u - \phi_v)}{R_{uv}} \).

A minimizer must have \( \frac{\partial \mathcal{E}(f)}{\partial \phi_v} = 0 \), and thus it is a voltage vector.

We use this theorem to calculate the change of effective resistance after deleting an edge.

**Lemma** Suppose \( f \) is an electric flow and \( e \) is an edge such that \( f_e + f_e \geq \beta \mathcal{E}(f) \).

The effective resistance \( \mathcal{R}' \) after removing an edge is at least \( \frac{\mathcal{R}}{1 - \beta} \) where \( \mathcal{R} \) is the original effective.

**Proof** Without loss of generality we assume that \( f \) is a flow from \( s \) to \( t \) of \( 1/R \) unit.

Using the previous theorem, \( \frac{1}{\mathcal{R}} = \min_{\phi} \sum_{uv} \frac{(\phi_u - \phi_v)^2}{R_{uv}} = \mathcal{E}(f) \).

Let \( \phi_e = x' y' \). Then \( \frac{(x' - y')}{} \) is at least \( \beta \mathcal{E}(f) \) by assumption.

By deleting the edge \( e \), it is the same as increasing \( R_e \) to infinity.
We use the above result to bound the number of deleted edges throughout the algorithm.

**Theorem.** For $p = O(m^{1/3})$, the number of edges deleted is $O(n^{4/3})$.

This implies that the total complexity is $O(n^{4/3})$.

Let $R(i)$ be the effective resistance at step $i$.

**Initial effective resistance**

Let $k^*$ be the optimal maximum flow between $s$ and $t$.

This implies that there is an $s-t$ cut with $k^*$ edges.

So one of the edges in the cut would have flow value at least $\frac{1}{k^*}$, when we send one unit of flow from $s$ and $t$.

Thus, $R(0) \geq \frac{1}{k^*}$.

**Lower Bound on $R(T)$**

When we delete an edge, it has a flow value at least $p$, and thus the energy on that edge is at least $p^2 e = \frac{p^2}{w_e + \frac{\omega(W)}{m}} \geq \frac{p^2 e w}{m} \geq \frac{p^2 e}{k^*(m)}$ (total energy), as total energy $\leq (1+\epsilon)W$.

By setting $\beta = \frac{p^2 e}{k^*(m)}$ and apply the lemma, the effective resistance increases by a factor of $\left(1 - \frac{p^2 e}{k^*(m)}\right)^{-1}$ after we delete an edge.

So, if we have deleted $h$ edges throughout the algorithm, then $R(T) \geq R(0) \cdot \left(1 - \frac{p^2 e}{k^*(m)}\right)^{-h} \geq \frac{1}{k^*} \left(1 - \frac{p^2 e}{k^*(m)}\right)^{-h}$

**Upper Bound on $R(T)$**

The energy of the flow of $k$ units is at most $(1+\epsilon)W(T)$.

(Here it is not precise because some edges have been removed. But we can ensure that the number of edges removed is at most a small fraction of $k^*$. Thus the energy of the flow is only slightly increased, by scaling up the flow on the remaining edges slightly. So the total energy would not be much higher than that.)

Thus, $R(T) \leq \frac{(1+\epsilon)W(T)}{k^*}$, by scaling down the flow value by a factor of $k$.

Note that $W(T) = \sum_{T} w(T) (1+\frac{\omega(T)}{m})$ by the multiplicative update rule.
Note that \[ W(t+1) = \sum_{t} w_t e_t (1 + \frac{e_t}{p} f_t e_t) \] by the multiplicative update rule.

\[ = \frac{1}{2} \sum_{t} w_t e_t + \frac{e_t}{p} \sum_{t} w_t f_t e_t \leq \text{wlex}_1 \left( e_t \frac{E(t)}{p} \right). \]

So, \[ W(t) \leq m \cdot \left( 1 + \frac{E(t)}{p} \right)^T \leq m \cdot 2 \frac{E(t)}{p} \leq m \cdot 2 \frac{2\rho n}{k^2} \text{ as } T = \frac{\rho n}{k}. \]

Therefore, \[ R(t) \leq \frac{(2\rho n) \cdot \frac{2\rho n}{k^2}}{k^2}. \]

**Putting together**

Combining the upper and lower bound on \( R(t) \), we have \[ \frac{1}{k^2} \left( 1 - \frac{2\rho n}{m(\rho n)} \right)^h \leq \frac{(2\rho n) \cdot 2 \frac{2\rho n}{k^2}}{k^2}. \]

Using \( k^2/k \leq m \), this implies \[ h \leq \frac{\ln(\frac{1}{m})}{\ln(\frac{2\rho n}{\rho n})} + h \ln \left( 1 - \frac{2\rho n}{m(\rho n)} \right) \]

\[ \Rightarrow h \leq \frac{\ln(\frac{2\rho n}{m(\rho n)})}{\ln(1 - \frac{2\rho n}{m(\rho n)})} \]

\[ \Rightarrow h \leq \frac{2\rho n}{\frac{2\rho n}{m(\rho n)}} \]

Using \( \ln(1) < c \) for \( c \in (0,1) \).

\[ \Rightarrow h \leq \frac{m(\rho n)}{2\rho n} \]

Setting \( p = \tilde{O}(m^{1/2}) \), we have \( h = \tilde{O}(m^{1/2}) \), as required.

**Remarks**

- The algorithm can be generalized to work for the capacitated case; see [4].
- There is an \( \tilde{O}(m^{1/2}) \) algorithm for approximating minimum s-t cut; see [4].
- Using Laplacian solver to solve linear program may have wider applications, for example it is known that for some min-cost flow problem the linear systems in the interior point algorithms are Laplacian.
- This approach is not known to apply for directed graphs. Can you do it?

**References**

Concluding Remarks

Through this course we have seen various applications of spectral techniques in design and analysis of algorithms. It has become a more important technique in recent years, with significant progress in designing better approximation algorithms and faster algorithms.

It would be very nice if you could use spectral techniques to prove new theorems or apply these results in new applications, but I do not expect everyone to do that.

Rather, I hope this course to be useful in:

- learning basic concepts in linear algebra, e.g. eigenvalues, singular values, matrix norms, etc.
- learning useful tricks, e.g. Cauchy-Schwartz, randomized rounding, concentration inequalities, etc.
- learning general techniques, e.g. random walk, iterative methods, multiplicative update, etc.
- learning interesting objects, e.g. random graphs, expander graphs, electric flow, Chebyshev polynomials, etc.
- learning famous results, e.g. Cheeger’s inequality, expander codes, conjugate gradient, etc.
- learning problem solving skills, e.g. algebraic formulations, reduction, approximation, recursion, etc.

I hope you will find these linear algebraic techniques useful in reading papers and solving problems.

I would be happiest if you would later tell me that this course is useful in your research.

Thanks everyone for taking this course, especially those who come to Shaw College every week!

(If there is more time, I would like to talk about graph drawing and graph isomorphism.)

---

Remaining Schedule

Apr 26: homework 3 posted.

May 8: homework 3 due, 5pm, Dropbox outside 924

May 18: project report, 11:59 pm, no late report accepted, by email to chi@eee.cuhk.hk

THE END