CS 466/666 Algorithm Design and Analysis . Spring 2020

Lecture 22 : Maximum flow and Laplacian solvers

We complete our journey by seeing how Laplacian solvers can be used to design faster algorithms for maximum flow.

Then we discuss fast Laplacian Solvers, and review and preview what we have and have not covered in this course.

Maximum flow in undirected graphs

In this problem, we are given an undirected graph $G=(V,E)$, where each edge e has a capacity c_e,

a source vertex s and a sink vertex t.

The objective is to find a maximum s-t flow subjected to capacity constraints and flow conservation constraints.

More formally, we have two variables f_{uv} and f_{vu} for each edge uv.

The following is the linear programming relaxation of the problem.

$$\max \quad f(\delta^{out}(s)) - f(\delta^{in}(s))$$

$$\text{st.} \quad f(\delta^{out}(v)) = f(\delta^{in}(v)) \quad \text{for all } v \in V-\{s,t\} \qquad // \text{ flow conservation constraints}$$

$$f_e \leq c_e \quad \text{for all } e\in E \qquad // \text{ capacity constraints}$$

$$f_e \geq 0 \quad \text{for all } e\in E,$$

where $\delta^{in}(v)$ and $\delta^{out}(v)$ are the set of incoming and outgoing edges of v, and $f(F)=\sum_{e\in F} f(e)$ for $F\subseteq E$.

For simplicity, we assume $c_e=1$ for all edges.

So, the optimal value of the linear program is in $[0,m]$ where m is the number of edges.

We can reduce the optimization problem to $O(\log m)$ decision problems, by doing binary search on the

objective value and by replacing the objective function by the constraint $f(\delta^{out}(s)) - f(\delta^{in}(s)) = k$.

Multiplicative weight update method

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

We will think of the objective constraint $\left(f(\delta^{out}(s)) - f(\delta^{in}(s)) = k \right)$, the non-negative constraints $(f_e \geq 0, \forall e \in E)$,

and the flow conservation constraints $\left(f(\delta^{out}(v)) = f(\delta^{in}(v)) \ \forall v \in V-\{s,t\} \right)$ as the "easy" constraints,

and we will design a polynomial time oracle to satisfy them with small width.

We will think of the capacity constraints $(x_e \leq 1 \ \ \forall e\in E)$ as "hard" constraints,

and we will use the multiplicative weight update method to "combine" them and satisfy them approximately.

With this setup, by the analysis of the multiplicative weight update method we have seen last time, we can find

an almost feasible solution $(f_e \leq 1+\varepsilon \ \ \forall e\in E)$ if we could solve $O(\frac{\rho \ln n}{\varepsilon^2})$ Subproblems of the following form.

① f satisfies the objective constraint, the non-negative constraints, and the flow conservation constraints.

② f approximately satisfies the multiplicative weight update constraint $\sum_e w_e f_e \leq (1+\varepsilon) \sum_e w_e$.

Following L21, we should satisfy $\sum_e w_e f_e \leq \sum_e w_e$, but it is easy to check that it is okay to respond with an approximate feasible solution, as we are just designing approximation algorithms anyway.

③ $f_e \leq \rho \quad \forall e \in E$ where ρ is the width of the oracle that controls the convergence rate.

If these subproblems can always be solved (for different values of w_e), then we have an oracle of small width.

Then it follows from the result in L21 that the average solution of these subproblems would be an almost feasible solution, with $f_e \leq 1+\varepsilon$ for all $e \in E$.

To obtain a feasible solution with $f_e \leq 1 \quad \forall e \in E$, we can simply scale down the solution by a factor of $1+\varepsilon$, then it will satisfy all constraints exactly, while the objective value becomes $\frac{k}{1+\varepsilon} \approx k(1-\varepsilon)$.

Remarks:

① In the result stated in L21, the convergence rate is $O(\frac{\rho^2 \ln n}{\varepsilon^2})$.

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

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

② If we would like to have an integral solution, then we can use the random walk algorithm in HW5 to round the fractional solution to an integral solution in near-linear time.

Electrical flow

In HW5, we use shortest paths to design an oracle for the multiplicative weight update method.

Interestingly, we can design a better oracle using electrical flow.

We would like to show that the electrical flow algorithm can be used to solve the subproblems with the following four properties, when it is given w_e on each edge $e \in E$.

① It 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 $\sum_e w_e f_e \leq (1+\varepsilon) \sum_e w_e$.

③ When the maximum flow problem is feasible, the oracle can always return f with $f_e \leq O(\sqrt{m})$ for all $e \in E$.

④ The oracle can be implemented in $\tilde{O}(m)$ time.

Assuming this can be done, we can find a $(1-\varepsilon)$-approximate solution in $O(\frac{\sqrt{m} \ln n}{\varepsilon^2})$ iterations, and the total running time would be $\tilde{O}(\frac{m^{1.5}}{\varepsilon^2})$ thanks to ④.

Oracle We will use an electrical flow algorithm for the oracle.

Oracle We will use an electrical flow algorithm for the oracle.

Given w_e on each edge, we set the resistance of edge e to be $r_e = w_e + \frac{\varepsilon W}{m}$, where $W = \sum_e w_e$.

Then, we compute the electrical flow f that sends k units of electricity from s to t, with resistance r_e on each edge e. This is the oracle, to return this flow f.

Algorithm So, the whole algorithm is pretty simple and interesting.

Initially, every edge has the same resistance, and we compute the k-unit s-t electrical flow.

In each iteration, we update the weight by setting $w_e^{t+1} = w_e^t \left(1 + \frac{\varepsilon}{\rho} f_e^t\right)$ and update r_e accordingly, and then compute another k-unit s-t electrical flow with the updated resistances.

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.

After $O\left(\frac{\rho \ln n}{\varepsilon^2}\right)$ iterations, we take the average of the flows in these iterations and it is a good solution.

Analysis

We check the four properties one by one.

① It should be clear by construction that the flow constraints, the non-negative constraints, and the objective value constraint are all satisfied exactly.

② Here we use an energy argument.

If the flow problem is feasible, then there is a flow $\tilde{f}$ of k units from s to t without violating the capacity constraints, so that $\tilde{f}_e \le 1$ for all $e \in E$.

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

Recall from L15 that the electrical flow minimizes the energy, so we must have $\sum_e f_e^2 r_e \le (1+\varepsilon) W$, or otherwise we can conclude that the flow problem is infeasible

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

$$\left(\sum_e w_e f_e\right)^2 \le \left(\sum_e w_e f_e^2\right)\left(\sum_e w_e\right) \le \left(\sum_e r_e f_e^2\right)\left(\sum_e w_e\right) \le (1+\varepsilon) W^2.$$

This implies that $\sum_e w_e f_e \le (1+\varepsilon) \sum_e w_e$, as promised.

③ Clearly, the energy on one edge cannot be more than the total energy,

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

④ It is a seminal result by Spielman and Teng that Laplacian equations can be solved in $\tilde{O}(m)$ time, and this implies a near-linear time algorithm for computing electrical flows as shown in L15.

We will discuss more about Laplacian solvers later.

This completes the $\tilde{O}(m^{1.5})$ time algorithm to give an $(1-\varepsilon)$-approximation algorithm for computing maximum flow in undirected graphs for constant ε, matching the best known combinatorial algorithms in this regime.

This approach can be improved to $\tilde{O}(m^{4/3}/\varepsilon^2)$ time by slightly modifying the algorithm and improving the analysis using more careful arguments about electrical flows.

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.

Wrapping up

In the lecture and slides, I will discuss two simple and elegant algorithms for Laplacian solvers.

One based on a simple cycle fixing procedure for computing electrical flow, and another based on Gaussian elimination with sparsification. See the references below.

Then, I will review and highlight what we have covered in this course, and mention something interesting that we have not covered.

Thanks for taking the course. Hope you enjoy it and will find some of it useful in the future.

References

- Christiano, Kelner, Madry, Spielman, Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs, 2010.
- 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.