A Theorist's Toolkit **(CMU 18-859T, Fall 2013)**

Lecture 14: Linear Programming II

October 23, 2013

Lecturer: Ryan O'Donnell *Scribe: Stylianos Despotakis*

1 Introduction

At a big conference in Wisconsin in 1948 with many famous economists and mathematicians, George Dantzig gave a talk related to linear programming. When the talk was over and it was time for questions, Harold Hotelling raised his hand in objection and said "But we all know the world is nonlinear," and then he sat down. John von Neumann replied on Dantzig's behalf "The speaker titled his talk 'linear programming' and carefully stated his axioms. If you have an application that satisfies the axioms, well use it. If it does not, then don't." [Dan02]

Figure 1: A cartoon Dantzig's Stanford colleagues reported as hanging outside his office [Coo11]

In this lecture, we will see some examples of problems we can solve using linear programming. Some of these problems are expressible as linear programs, and therefore we can use a polynomial time algorithm to solve them. However, there are also some problems that cannot be captured by linear programming in a straightforward way, but as we will see, linear programming is still useful in order to solve them or "approximate" a solution to them.

2 Maximum Flow Problem

Max (s-t) Flow Problem is an example of a true linear problem. The input is a directed graph $G = (V, E)$ with two special vertices, a "source" $s \in V$, and a "sink" $t \in V$. Moreover,

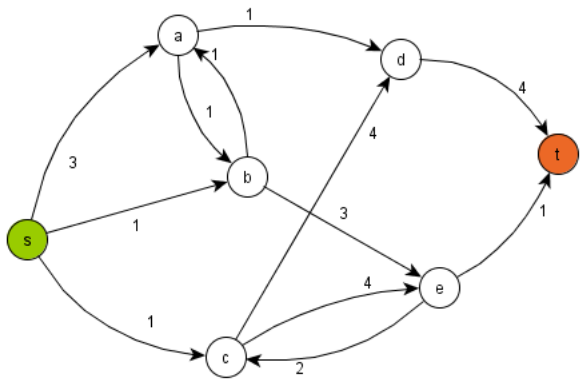

Figure 2: An example of an input to the Max Flow problem

each edge $(u, v) \in E$ has a "capacity" $c_{uv} \in \mathbb{Q}^{\geq 0}$. In Figure 2, we can see an example of such graph. Capacities represent the maximum amount of flow that can pass through an edge. The objective of the problem is to route as much flow as possible from s to t [Wik13a].

The maximum flow problem was first studied in 1930 by A.N. Tolstoy̌ as a model of Soviet railway traffic flow [Sch02] (e.g. nodes are cities, edges are railway lines, there is a cement factory at s, there is a plant that needs cement at t, and capacities represent how much cement a railway can ship in one day in some units).

For every node $v \notin \{s, t\}$, the incoming flow to v must be equal to the outgoing flow from v. This is called *flow conservation* constraint. Considering this, we can express the problem as a linear program as follows:

$$\begin{aligned} \max_{f} \quad & \sum_{v:s\to v} f_{sv} - \sum_{u:u\to s} f_{us} \\ \text{s.t.} \quad & \sum_{u:u\to v} f_{uv} = \sum_{w:v\to w} f_{vw}, \quad \forall v \neq s, t \qquad \text{(flow conservation)} \\ & 0 \leq f_{uv} \leq c_{uv}, \qquad \forall (u, v) \in E \quad \text{(capacity constraints)} \end{aligned}$$

Now, it's easy to see that any feasible solution to this program is a feasible flow for the graph. In Figure 3, we can see an optimal solution to the example of Figure 2, where the labels in the edges are of the form f_{uv}/c_{uv}.

The fact that Max Flow can be expressed as a linear program, tells us that it can be solved in polynomial time. Not only that, but since there are efficient algorithms for linear programming, it can also be solved efficiently.

We also mention that there are many more efficient algorithms for Max Flow, like for example the Ford–Fulkerson algorithm which is more combinatorial in nature, but the approach above shows us better the power of linear programming.

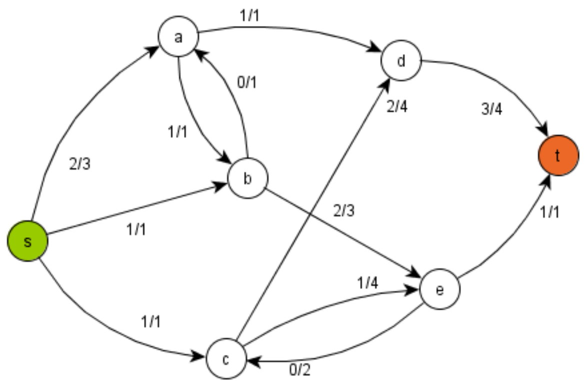

Figure 3: An optimal solution to the example of Figure 2

3 Maximum Perfect Matching in Bipartite Graphs

This problem can be solved using the Max Flow problem, but for illustrative purposes, we will see a different approach here.

In this problem we have a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$, and each edge $\{u, v\} \in E$ has a weight $w_{uv} \in \mathbb{Q}$. The objective is to find a perfect matching of maximum weight. Without loss of generality, we can assume that a perfect matching exists in the graph, because if one doesn't exist we can add some edges with weigths $-\infty$ (or 0 if all the weights are non-negative), in order to create a perfect matching.

We can imagine the vertices in U as n people, the vertices in V as n jobs, and the weight of an edge $\{u, v\}$ as the ability of person u to do the job v. Our goal is then to assign one job to each person in the best possible way.

This problem cannot directly transformed to a linear program. But instead, we can write it as an *integer linear program* (ILP). An integer linear program is like a linear program (LP), but with some additional constraints that some of the variables are integers.

Suppose we have an indicator variable x_{uv} for every edge $\{u, v\} \in E$, such that $x_{uv} = 1$ if $\{u, v\}$ belongs to the perfect matching, and $x_{uv} = 0$ otherwise. Then, we can write the problem as an integer linear program (ILP) as follows:

$$\begin{aligned}
\max_x \quad & \sum_{\{u,v\}\in E} w_{uv} x_{uv} \\
\text{s.t.} \quad & \sum_{u\sim v} x_{uv} = 1, \quad \forall u \in U && \text{(every person has a job)} \\
& \sum_{u\sim v} x_{uv} = 1, \quad \forall v \in V && \text{(every job is assigned to someone)} \\
& x_{uv} \in \{0, 1\}, \quad \forall \{u, v\} \in E && \text{(integer constraints)}
\end{aligned}$$

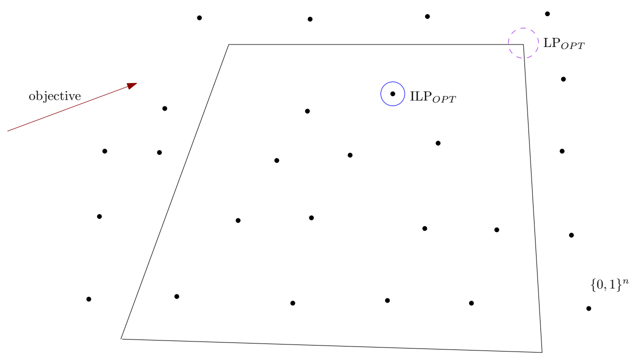

Figure 4: The dots are integers points of $\{0,1\}^n$, and inside the convex polytope are all the feasible points of the LP

Now suppose that in the above program we drop the integer constraints of the form $x_{uv} \in \{0,1\}$, and we replace them with the linear constraints $0 \leq x_{uv} \leq 1$, $\forall \{u,v\} \in E$. Then, we get a linear program (LP), which can solve in polynomial time. This procedure is called a *relaxation* of the integer program.

Relaxation Facts:

- If the LP is infeasible, then the ILP is infeasible.
- If the LP is feasible, then either the ILP is infeasible, or the ILP is feasible and LP$_{OPT} \geq$ ILP$_{OPT}$ (in case of a maximization problem).

The above two facts hold because as we can see from the corresponding programs, every feasible solution to the ILP is a feasible solution to the LP.

The second fact is useful because it gives an upper bound to the optimal solution we are searching for. Therefore, one approach is to use some heuristic to solve our problem, and if the objective is close to the other bound, then we know that we are "near" the optimal solution.

However, in this case, even more is true, which are not true in general for every relaxation. More specifically, we have the following theorem.

Theorem 3.1. *All extreme points of the LP are integral.*

Proof. We will prove the contrapositive, if $\widetilde{x}$ is a feasible, non-integral solution to LP, then it is not an extreme point, i.e. we can write $\widetilde{x} = \frac{1}{2}x^+ + \frac{1}{2}x^-$, for two distinct feasible solutions x^+ and x^-.

Since $\widetilde{x}$ is non-integral, there is some $\widetilde{x}_{uv}$ that is non-integral. The sum of the edges incident to v is 1, therefore there is at least one other edge incident to v that is non-integral, say $\{v,z\}$. Then, look at vertex z. Again there is one other edge incident to z that is non-integral. This procedure can't go on forever, therefore at some point we will end up with a cycle that returns to v.

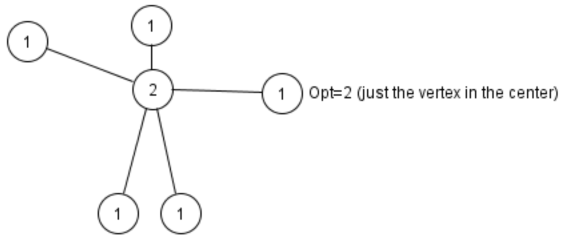

Figure 5: A graph where a greedy algorithm for Vertex Cover can perform very poorly. The greedy may return all the vertices with degree 1 with total cost 5, but the minimum vertex cover has cost 2.

The cycle is "non-integral", therefore $\exists \epsilon > 0$ such that $\epsilon < \widetilde{x}_{uv} < 1-\epsilon$, for every $\{u, v\}$ on the cycle. The graph is bipartite, therefore the cycle is even. Add ϵ to all odd edges along the cycle, and $-\epsilon$ to all even edges. The sum of the edges along the cycle remains the same, therefore you get a feasible solution x^+. Similarly, if you add ϵ to all even edges along the cycle, and $-\epsilon$ to all odd edges, you get a feasible solution x^-. The solutions x^+ and x^- are distinct, since $\epsilon > 0$, and it holds that $\widetilde{x} = \frac{1}{2}x^+ + \frac{1}{2}x^-$, because when we add something in an edge in x^+, we subtract the same thing in x^-. □

Observation 3.2. *At least one of x^+, x^- in the proof has objective at least as good as $\widetilde{x}$.*

Suppose you solve the LP with an LP solver, which gives an optimal solution which is non-integral (it is not a vertex of the polytope). The proof above gives an algorithm to convert this solution to an optimal integer solution. Therefore, the theorem implies that the problem can be solved in polynomial time.

Fact 3.3. *The "integrality" property of the theorem holds for any linear program of the form:*

$$\begin{aligned} \max_x / \min_x \quad & c \cdot x \\ s.t. \quad & b' \le Mx \le b, \\ & \ell \le x \le u, \end{aligned}$$

where b', b, ℓ, u are integer vectors (possibly $\pm\infty$), and M is totally unimodular. A matrix is called totally unimodular if all of its square submatrices have determinant $+1, -1,$ or 0.

4 Minimum Vertex Cover

In the Min Vertex Cover problem, the input is a graph $G = (V, E)$ with a cost $c_v \ge 0$ for every vertex $v \in V$. A *vertex cover* is a set of vertices $S \subseteq V$ such that all edges in E have at least one endpoint in S. The goal is to find a vertex cover S in the graph that minimizes the quantity $c(S) = \sum_{v \in S} c_v$.

The Min Vertex Cover problem is NP-hard. Therefore, we can't expect to solve it to optimality using linear programming. However, as before we can express it as an integer linear program. Let x_v be an indicator variable that is 1 if v belongs to the vertex cover, and 0 otherwise. Then, we can write the problem as follows:

$$\begin{array}{rll} \min\limits_{x} & \sum_{v\in S} c_v x_v & \\ \text{s.t.} & x_u + x_v \geq 1, & \forall\{u,v\} \in E \quad \text{(every edge is covered)} \\ & x_v \in \{0,1\}, & \forall v \in V. \end{array}$$

As before, we can relax this integer program by replacing the constraints $x_v \in \{0,1\}$ with the constraints $0 \leq x_v \leq 1$, to get a linear program (LP). We know that $\text{LP}_{OPT} \leq OPT$. Suppose we solve the LP to optimality, and we get back an optimal LP-feasible solution x^* (often called "fractional solution").

Example 4.1. *Let G be a K_3 with three vertices u, v, w, and costs $c_u = c_v = c_w = 1$. The optimal vertex cover consists of two vertices, therefore $OPT = 2$. The optimal fractional solution is to assign $\frac{1}{2}$ to every vertex, i.e. $x_u = x_v = x_w = \frac{1}{2}$, therefore $LP_{OPT} = \frac{3}{2}$.*

In general, if G was a clique K_n, the optimal vertex cover would have cost $OPT = n-1$, but $LP_{OPT} \leq \frac{n}{2}$, since we can always assign $\frac{1}{2}$ to every vertex. This means that there is a gap of about 2 between the optimal integer solution and the optimal fractional solution.

LP Rounding is called the procedure that takes an LP-feasible solution and somehow converts it to an actual ILP-feasible solution with almost as "good quality" as the LP-feasible solution.

For the Vertex Cover problem, say $\widetilde{x}$ is a feasible LP solution. Define $S = S_{\widetilde{x}} = \left\{v : \widetilde{x}_v \geq \frac{1}{2}\right\}$.

Fact 4.2. *S is a valid vertex cover.*

Proof. For every edge $\{u,v\} \in E$, it holds that $x_u + x_v \geq 1$. This means that at least one of x_u and x_v is at least $\frac{1}{2}$, therefore at least one of u and v belongs to S. □

Fact 4.3. *$cost(S_{\widetilde{x}}) \leq 2 \cdot cost_{LP}(\widetilde{x})$*

Proof. It holds that $\text{cost}_{LP}(\widetilde{x}) = \sum_{v\in V} c_v \widetilde{x}_v \geq \sum_{v\in S} c_v \widetilde{x}_v \geq \sum_{v\in S} \frac{1}{2} c_v = \frac{1}{2}\text{cost}(S)$ $(\geq \frac{1}{2} OPT)$. □

Therefore, we have that

$$\frac{1}{2} OPT \leq \text{LP}_{OPT} \leq OPT,$$

and the first inequality is the best possible due to the clique we've seen in Example 4.1.

The above rounding procedure gives a poly-time algorithm that finds a solution with value at most $2\text{LP}_{OPT} \leq 2OPT$. Therefore, this is a 2-approximation algorithm for Vertex Cover. It is also known that if the unique games conjecture is true, then Vertex Cover cannot be approximated within any constant factor better than 2 [Wik13b].

5 Duality

Suppose you have a linear program of the form:

$$\begin{array}{ll} \max\limits_{x} & c \cdot x \\ \text{s.t.} & a^{(1)} \cdot x \le b_1, \\ & a^{(2)} \cdot x \le b_2, \\ & \ldots\ldots \\ & a^{(m)} \cdot x \le b_n. \end{array}$$

Let $\lambda_1, \lambda_2, \ldots, \lambda_m \ge 0$ be numbers such that if you multiply the i^{th} constraint with λ_i and you add all the constraints together, you get $c \cdot x \le \beta$, for some number β. Then, you know that β is an upper bound to the optimal solution of the linear program. Therefore, you want to find λ_i's that achieve the minimum possible β. In other words, you want to solve the following linear program:

$$\begin{array}{ll} \min\limits_{\lambda} & \sum\limits_{i=1}^{m} b_i \lambda_i \\ \text{s.t.} & \lambda_1 a_{11} + \lambda_2 a_{21} + \ldots + \lambda_m a_{m1} = c_1, \\ & \lambda_1 a_{12} + \lambda_2 a_{22} + \ldots + \lambda_m a_{m2} = c_2, \\ & \ldots\ldots \\ & \lambda_1 a_{1n} + \lambda_2 a_{2n} + \ldots + \lambda_m a_{mn} = c_n, \\ & \lambda_1, \lambda_2, \ldots, \lambda_m \ge 0. \end{array}$$

This is called the *dual* linear program. Farkas Lemma, we've seen in the previous lecture, implies that if the original linear program (the *primal*) is feasible, then the dual is feasible, and the two LP's have the same value (which may be $\pm\infty$).

Rule of Life: If you have an LP, you should take its dual and try to "interpret" it.

To see an example of what this means, let's go back to the Max Flow problem. If we take its linear program and clean it up a little bit (e.g. if there is a constraint with something $\le b$ and another one with the same thing $\ge b$, we convert it to a constraint of the form something $= b$), we get the following dual linear program:

$$\begin{array}{lll} \min\limits_{\lambda,\mu} & \sum\limits_{(u,v)\in E} c_{uv} \lambda_{uv} & \\ \text{s.t.} & \mu_s = 1, & \\ & \mu_t = 0, & \\ & \lambda_{uv} \ge 0, & \forall (u,v) \in E \\ & \lambda_{uv} \ge \mu_u - \mu_v, & \forall (u,v) \in E, \end{array}$$

where the variables λ_{uv} correspond to the capacity constraints of the Max Flow, and the variables μ_v correspond to the flow conservation constraints.

This dual linear program is the natural LP relaxation of the ILP for the Min (s-t) Cut problem. In this problem, we want to find a set of vertices $S \subseteq V$ such that $s \in S$ and

$t \notin S$, which minimizes the quantity $\sum_{u \in S, v \notin S} c_{uv}$. The variables μ_v are indicator variables that say if v belongs to S or not, and the variables λ_{uv} are indicator variables that say if the edge (u, v) goes from the set S to the set $V \setminus S$ or not.

From the observation above, we conclude that

$$\text{Opt Max } s\text{-}t \text{ Flow} = \text{Opt Min fractional } s\text{-}t \text{ Cut} \leq \text{Opt Min } s\text{-}t \text{ Cut}.$$

It turns out that the inequality is actually an equality, which means that the Min Cut problem is also solvable in polynomial time.

References

[Coo11] William Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011.

[Dan02] George B Dantzig. Linear Programming. *Operations Research*, pages 42–47, 2002.

[Sch02] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

[Wik13a] Wikipedia. Maximum flow problem. http://en.wikipedia.org/wiki/Maximum_flow_problem, 2013. [Online; accessed 27-October-2013].

[Wik13b] Wikipedia. Vertex cover. http://en.wikipedia.org/wiki/Vertex_cover, 2013. [Online; accessed 27-October-2013].