Algorithmic Methods 15.11.2010

Lecture Notes 5: Applications of Linear Programming

Professor: Yossi Azar *Scribe:Ran Roth*

1 Introduction

This lecture contains the following topics:

- Statement and proof of theorem D
- Linear programs with exponential number of constraints
- Approximation algorithms - the basics

2 Statement and proof of theorem D

We have used theorem D in the analysis of the ellipsoid algorithm. The term theorem D is used here because it is of the same characteristics as theorems A,B and C that were proven in the first lecture. We now close the gap by phrasing and proving this theorem.

Theorem 2.1. *Let $P = \{x : Ax \geq b\}$ be a polytope. Every $x \in P$ can be expressed as a convex combination of at most $n+1$ vertices of P.*

Proof. Let $x \in P$. We show a more general claim: if x tightly fulfills r independent constraints, then x can be expressed as a convex combination of at most $n+1-r$ vertices. Substituting $r = 0$ we get the theorem.

We prove the claim by induction on r, where the basis is $r = n$ and we decrease r in each step. The basis of the induction is $r = n$ and thus n constraints are fulfilled tightly. By theorem A, x is a vertex and thus can trivially be expressed as a combination of at most one vertex (itself).

We assume that x tightly fulfills r independent constraints. Writing these constraints as equalities we get a smaller polytope $P' \subseteq P$ such that $x \in P'$. By theorem B we know that every polytope has a vertex, let v be an arbitrary vertex of P'. Note that vertices of P' are also a vertices of P since a vertex is defined by any n tight constraints (according to theorem A), thus v is also a vertex of P.

We now consider the ray starting at v and containing the segment that connects x and v:

$$\forall t \geq 0, \quad \phi(t) = v + t(x - v)$$

Let t_0 be the maximal t such that $\phi(t)$ is still feasible (Note that $t_0 > 1$ since $x \in P'$ and P' is convex). Denote $y = \phi(t_0)$. Then y fulfills tightly the r original constraints plus one more, and by the inductive assumption it can be expressed as a convex combination of at most $n + 1 - (r + 1) = n - r$ vertices of P. Denote this combination by

$$y = \sum_{i=1}^{n-r} \lambda_i v_i \qquad \lambda_i \geq 0, \sum \lambda_i = 1$$

Then,

$$x = \frac{1}{t_0}y + (1 - \frac{1}{t_0})v = \sum_{i=1}^{n-r} \frac{\lambda_i}{t_0} v_i + (1 - \frac{1}{t_0})v$$

We have expressed x as a convex combination of $n - r + 1$ vertices of P. □

Corollary 2.2. *A polytope P is the convex hull of its vertices.*

2.1 Application: stochastic matrices

We define two types of matrices: doubly stochastic matrices and permutation matrices. We then show that every doubly stochastic matrix can be expressed as a convex combination of permutation matrices.

Definition 2.3. A matrix A is called a *doubly stochastic matrix* if

- All the entries are non-negative.
- The sum of every row is 1.
- The sum of every column is 1.

Definition 2.4. A matrix P is called a *permutation matrix* if it can be obtained by permuting the columns of the unity matrix.

Theorem 2.5. *A matrix A is doubly stochastic iff it can be expressed as a convex combination of permutation matrices.*

Proof. Permutation matrices are in particular doubly stochastic. In addition, a convex combination of doubly stochastic matrices is a doubly stochastic matrix. Thus, if A is a combination of permutation matrices, it is doubly stochastic.

In the non-trivial direction we show that every doubly stochastic matrix can be expressed as a convex combination of permutation matrices. Let us denote by P the polytope of doubly stochastic matrices. It is defined by the following constraints:

$$\sum_{j=1}^{n} \alpha_{ij} = 1 \tag{1}$$

$$\sum_{i=1}^{n} \alpha_{ij} = 1 \tag{2}$$

$$\alpha_{ij} \geq 0 \tag{3}$$

Note that one constraint is redundant in (1) and (2), since the sum of all rows equals the sum of all columns. Thus, we have in total $n^2 + 2n - 1$ constraints. Using the corollary of theorem D, it suffices to prove the claim only for the vertices of the polytope. Because every point in the polytope is a convex combination of its vertices, this proves the general claim. We show this by induction on n, the dimension.

The basis $n = 1$ is trivial, since the only stochastic scalar is 1 and it's also a permutation matrix. Let $A_{n\times n}$ be a vertex of P. Every matrix has n^2 entries (or variables), so by

theorem A, at a vertex exactly n^2 constraints are tightly fulfilled by A. Hence, we have at least $n^2 - 2n - 1$ constraints of the type $\alpha_{ij} = 0$. This means that on average, a row has at least $(n^2 - 2n + 1)/n \geq n - 2$ zero entries. Therefore, there exists a row that has at least $n - 1$ zeros. Let i be the index of this row and let j be the index of the single non-zero entry. Since A is doubly stochastic, the ith row and column have all zeros, except for the entry (i, j) which must be 1. We can now remove the ith row and jth column to obtain a matrix A' of dimension $(n-1) \times (n-1)$. By the inductive assumption, we can express A' as a convex combination of permutation matrices

$$A' = \sum_k \lambda_k P'^k_{(n-1)\times(n-1)}$$

We can now use the same convex combination $\{\lambda_i\}$ with P_i

$$A = \sum_k \lambda_k P^k_{(n)\times(n)}$$

where P^k are permutation matrices of dimension $n \times n$ obtained from P'^k by inserting the ith row and jth column with 1 in the entry (i, j) and all zeros. □

Corollary 2.6. *The vertices of P are the permutation matrices.*

3 Linear programs with exponential number of constraints

Quite surprisingly, the ellipsoid algorithm can be used to solve in polynomial time some linear programming problems having an exponential number of constraints. The main observation is that the algorithm needs to go over the constraints only in order to check the feasibility of the center of gravity of the current simplex. It turns out, that in some problems this can be done efficiently, without explicitly checking all the constraints. This is modelled by a *separation oracle*. The oracle is given as an input a point in space and returns in polynomial time whether or not it is feasible. In addition, if the point is not feasible, the oracle returns a contradicting constraint. Assuming we have such an oracle, we can solve the linear program in polynomial time, regardless of the number of constraints. Formally, one still needs to show how to transform the LP from LI to LSI without going over all the constraints, but this is a technical matter that can be solved quite easily.

3.1 Example 1

Problem definition. Let $G = (V, E)$ be a graph, and let $s, t \in V$. We would like to assign weights to the edges of G such that all the paths from s to t have at least a weight of 1. Our goal is to assign the weights such that the total weight is minimal. Phrasing this as a decision problem (instead of maximization problem), can we assign the weights such that all the paths from s to t have at least a weight of 1, but the total weight is at most α?
Solution. Let's write the problem in terms of LP. Define x_e to be the weight of edge $e \in E$. Let $\mathcal{P}$ be the set of all paths in G. It's easy to see that $\mathcal{P}$ can be exponentially large. We need to solve the feasibility of the linear program

1. $\sum_{e \in E} x_e \leq \alpha$
2. $\forall e \in E, \quad x_e \geq 0$
3. $\forall p \in \mathcal{P}, \quad \sum_{e \in \mathcal{P}} x_e \geq 1$

The number of constraints in 3 may be exponential. However, we can provide a separation oracle. Given a solution x, the oracle first checks explicitly if the constraints in 1 and 2 are satisfied. If not, it returns "infeasible" with the contradicting constraint. Otherwise, it finds the the shortest path from s to t on G with the weights given by x. If the shortest path has a weight smaller than 1, then x is not feasible and the shortest path is a contradicting constraint. Otherwise, x is feasible since all the paths have weight $\geq$ the weight of the shortest path ≥ 1.

3.2 Example 2

Problem definition. Let $G = (V, E)$ be a graph, and let $c : V \to \mathbb{R}_+$ be capacities assigned to the vertices. We would like to assign weights to the edges of G such that the weight of all edges incident to a vertex will never exceed its capacity. The goal is to assign the weights such that the weight of every spanning tree of G is at least β.

Solution. Define x_e to be the weight of edge $e \in E$. Let $\mathcal{T}$ be the set of all spanning trees of G, note that $\mathcal{T}$ can be exponentially large. We need to solve the feasibility of the linear program

1. $\forall e \in E, \quad x_e \geq 0$
2. $\forall u \in V, \quad \sum_{e \in (u,v)} x_e \leq w_u$
3. $\forall T \in \mathcal{T}, \quad \sum_{e \in T} x_e \geq \beta$

The number of constraints in 3 may be exponential. However, we can provide a separation oracle. Given a solution x, the oracle first checks explicitly if the constraints in 1 and 2 are satisfied. If not, it returns "infeasible" with the contradicting constraint. Otherwise, it finds a minimal spanning tree of G with the weights given by x. If the MST that was found has a weight smaller than β, then x is not feasible and the MST is a contradicting constraint. Otherwise, x is feasible since all the spanning trees have weight $\geq$ the weight of an MST $\geq \beta$.

3.3 Example 3

Problem definition. Let $G = (V, E)$ be a graph, and let $c : E \to \mathbb{R}_+$ be capacities assigned to the edges. We would like to assign weights to the vertices of G such that the sum weights of both vertices incident to an edge will never exceed its capacity. The goal is to assign the weights such that even if we remove any $n/2$ vertices, the weight of the remaining vertices is at least γ.

Solution. Define x_v to be the weight $v \in V$. We need to solve the feasibility of the linear program

1. $\forall e = (u, v) \in E, \quad x_u + x_v \leq w_e$

2. $\forall u \in V, \quad x_u \geq 0$

3. $\forall S \subseteq V, |S| = \frac{n}{2} \quad \sum_{u \in S} x_u \geq \gamma$

The number of constraints in 3 is exponential. However, we can provide a separation oracle. Given a solution x, the oracle first checks explicitly if the constraints in 1 and 2 are satisfied. If not, it returns "infeasible" with the contradicting constraint. Otherwise, it simply sorts the weights given by x and calculates the sum of the $n/2$ smallest weights. If it is smaller than γ, then x is not feasible and the set with the smallest weights is a contraditcting constraint. Otherwise, x is feasible.

4 Approximation algorithms overview

Many interesting optimization problems in computer science are known to be NP-hard, thus an exact solution in polynomial time is not feasible, unless P=NP. One way to deal with this difficulty is to introduce the concept of approxmiation algorithms.

Definition 4.1. Let $\mathcal{I}$ be the set of inputs to some minimization problem. For $I \in \mathcal{I}$ we denote by $S(I)$ the feasible solutions to the problem. For every solution we have a different value of the goal (cost) function $c : S(I) \to \mathbb{R}_+$. Denote by $OPT(I) = \min c(S(I))$ the minimal (optimal) solution for input I. Let ALG be an algorithm for the problem, returning for an input I a feasible set whose value is $ALG(I)$. ALG is called a α-approximation algorithm if for every $I \in \mathcal{I}$ we have $\alpha OPT(I) \geq ALG(I)$.

We are typically interested in finding a polynomial time approximation algorithm with the best approximation ratio α. As long as the running time is polynomial, the interesting goal is usually to improve the approximation as much as possible. The question of whether we can improve the exact polynomial running time for a given approximatoin comes in second place.

4.1 Example: unweighted vertex cover

Let $G = (V, E)$ be a graph. $S \subset V$ is called a *vertex cover* of G if for every $(u, v) \in E$ we have $u \in S$ and/or $v \in S$. The goal is to find a vertex cover with minimal size.

We give the following simple AVC1 approximation algorthim.

Input: A graph $G = (V, E)$
Output: A set $S \subseteq V$ which is a vertex cover of G

1: $A \leftarrow \emptyset$
2: $E' \leftarrow E$
3: **while do**$E' \neq \emptyset$
4: Choose $(u, v) \in E'$
5: $A \leftarrow A \cup \{u\} \cup \{v\}$
6: $E' \leftarrow E' - \{(u, x)\}_{x \in V}$
7: $E' \leftarrow E' - \{(v, x)\}_{x \in V}$
8: **end while**
9: return A

Claim 4.2. *AVC1 is a 2-approximation algorithm to vertex cover.*

Proof. OPT must choose at least one vertex from every edge that AVC1 considered. Since for every such edge, AVC1 took both of its vertices, and these are all the vertices it took, we have $AVC1(I) \leq 2OPT(I)$. □

Note: In this case it can be shown that a greedy algorithm (taking each time the vertex with the highest number of incident edges) only achieves an approximation ratio of $log(n)$.

4.2 Example: weighted set cover

Let $G = (V, E)$ be a graph and let $w : V \to \mathbb{R}_+$ be a weight function on its vertices. The goal is to find a vertex cover S with minimal weight $w(S) = \sum_{u \in S} w(u)$.

We can use linear programming to obtain an approximation algorithm. We first write the problem as an ILP (Integer Linear Programming). For every $v \in V$, let x_v be an indicator variable, where 1 means we took the v in the vertex cover, and 0 means we didn't. The problem can be phrased as following

$$\begin{aligned} \min w(x) &= \sum w_i x_i \\ &\text{such that} \\ \forall (u,v) \in E, \quad x_u + x_v &\geq 1 \\ x_i &\in \{0,1\} \end{aligned}$$

Of course ILP is NP-hard (otherwise we would have been able to solve vertex cover which is known to be NP-hard). In order to use linear programming, we use a method called *relaxation*. We relax the constraint $x_i \in \{0,1\}$ to $0 \leq x_i \leq 1$ and obtain a normal fractional linear program that can be solved in polynomial time.

The ouput of the linear program x is at most OPT, since we only allowed more solutions to become feasible. However, we would like to receive an integral solution, while x may be fractional. We do this by using a rounding scheme. In this case the normal rounding turns out to be the best. Define

$$S = \{i : x_i \geq \frac{1}{2}\}$$

Claim 4.3. *S is a vertex cover and $w(S(I)) \leq 2OPT(I)$*

Proof. First, S is a vertex cover, since for every edge $(u, v) \in E$ we have $x_u + x_v \geq 1$. Thus, at least one of them is $\geq 1/2$ and belongs to S. In addition,

$$w(S) \leq \sum_{v \in S} w_v 2x_v \leq 2 \sum_{v \in V} w_v x_v \leq 2w(x) \leq 2OPT(I)$$

□