Lecture 8: Using the Dual

20th Feb, 2015

1 The Dual LP

Consider the following LP.

$$\begin{aligned} \texttt{lp} := \min \quad & c^\top x & (1) \\ \text{subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Consider the 'Lagrangean' function defined as

$$L(y) := \min_{x \geq 0} \quad c^\top x + y^\top (b - Ax)$$

Here y is a vector in $\Re^m$, where m is the number of non-trivial constraints. Two things are to be noted. First, for any y with $y_i \geq 0$ for all i, $L(y) \leq \texttt{lp}$ — this can be seen by plugging in x^*. Therefore, $\max_{y \geq 0} L(y) \leq \texttt{lp}$. Second, one can find a "closed form" for $L(y)$ thus.

$$L(y) = \begin{cases} y^\top b & \text{if } c^\top - y^\top A \geq 0 \\ -\infty & \text{otherwise.} \end{cases}$$

Therefore, $\max_{y \geq 0} L(y)$ can be expressed as another linear program. This is called the dual linear program.

$$\begin{aligned} \texttt{dual} := \max \quad & y^\top b & (2) \\ \text{subject to} \quad & y^\top A \leq c^\top \\ & y \geq 0 \end{aligned}$$

By the discussion above we get that $\texttt{dual} \leq \texttt{lp}$. This is known as weak duality. The strong duality theorem, which we will not cover in class, says that in fact $\texttt{dual} = \texttt{lp}$.

Complementary Slackness. Let (x, y) be optimum solutions to `lp` and `dual` respectively. Note that $x \in \Re^n$ and $y \in \Re^m$. Variables in the primal correspond to constraints in the dual, and vice versa. Complementary slackness says that if a primal variable is strictly positive in the optimal solution, then the corresponding dual constraint must be satisfied with equality.

$$\text{For all } 1 \le i \le n, \quad (y^\top A_i - c_i)x_i = 0.$$
$$\text{For all } 1 \le j \le m, \quad y_i(a_j^\top x - b_j) = 0.$$

To see this, consider the expression $(y^\top Ax - c^\top x)$. Since x is feasible, $Ax \ge b$, and since y is feasible, $y \ge 0$. Together, we get $y^\top Ax \ge y^\top b = c^\top x$, where the last equality follows from strong duality. So $y^\top Ax - c^\top x \ge 0$. However, $y^\top A \le c^\top$, and so $y^\top Ax - c^\top x \le 0$ as well. Together we get $y^\top Ax - c^\top x = 0$, which implies the first complementary slackness conditions. The second is obtained similarly. In fact, the above proof also gives the if x is feasible for the primal and y is feasible for the dual, then (x, y) are a pair of optimal solution for the primal-dual pair. Why?

2 Dual fitting and Primal Dual on Set Cover

Recall the LP for set cover

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} c(S_i)x_i & & (3)\\ \text{subject to} \quad & \sum_{i:j\in S_i} x_i \ge 1 & \forall \text{ elements } j & \\ & x \ge 0 & & \end{aligned}$$

We can write it's dual as follows. There is a variable y_j for every element j.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} y_j & & (4)\\ \text{subject to} \quad & \sum_{j\in S_i} y_j \le c(S_i) & \forall \text{sets } S_i & \\ & y \ge 0 & & \end{aligned}$$

We now illustrate two generic techniques which allow the dual to be used in design of approximation algorithms. The first is dual fitting. Here, we use the dual only to argue about the approximation factor of an algorithm. The second is the primal-dual methodology where the dual plays an important part in both the design and analysis of the algorithm.

Dual Fitting. I claim that we have already witnessed this method of analysing algorithms without knowing the name. Recall the greedy algorithm for the set cover problem. When one picks a set S_i which minimizes $c(S)/|S \cap X_i|$ among all sets S where X_i is the current set of uncovered elements, then set $y_j = c(S_i)/|S_i \cap X_i|$. Clearly, $\sum_j y_j$ is the cost of the algorithm. We now claim that $\sum_{j\in S} y_j \le H_{|S|} \cdot c(S)$ for all sets S. This implies that y/H_K, where K is the largest size of a set, is a feasible dual solution, and thus the cost of the algorithm is at most H_K times the objective function of a valid dual, and thus at most H_K times the LP.

The claim is similar to what we proved in Lecture 2. Order the elements of a set S in the order in which they get assigned y_j, and let this be $\{1, 2, \ldots, |S|\}$. By the greedy property, we have $y_1 \le c(S)/|S \cap X| = c(S)/|S|$, $y_2 \le c(S)/(|S| - 1)$, and so on. This implies what is needed.

Primal-Dual Algorithms. Here, we use the dual to design an algorithm. At this point, I'd like to remark that the algorithm wouldn't solve any LPs. Rather, it would come up with an integral primal solution and a feasible dual solution such that the dual objective value is not too far from the cost of the integral primal solution.

The PD algorithm for set cover is as follows.

1. Initially set $y_j = 0$ for all elements j. Initialize A, the set of uncovered elements, to the whole universe.

2. While all elements have been covered

   (a) Raise y_j of all elements in A at the 'same rate' till for some set S, the dual constraint $\sum_{j \in S} y_j \leq c(S)$ is satisfied with equality.

   (b) Pick set S in the cover and delete all elements of S from A. That is, $A = SA \setminus S$.

As promised, the algorithm ends up with a set cover $\mathcal{S}$ and a dual solution y. By definition, y is feasible. We now need to compare $\sum_{S \in \mathcal{S}} c(S)$ and $\sum_j y_j$.

$$\begin{aligned}\sum_{S \in \mathcal{S}} c(S) &= \sum_{S \in \mathcal{S}} \sum_{j \in S} y_j \\ &= \sum_{j=1}^{n} y_j \sum_{S \in \mathcal{S} : j \in S} 1 \\ &\leq f \sum_{j=1}^{n} y_j \leq f \cdot LP\end{aligned}$$

3 Solving LPs with exponentially many variables and polynomially many constraints

Last class we assumed we could solve the configuration LP which had exponentially many variables but only polynomially many constraints. I promised that we will see how this is (theoretically) possible in this class using duality. However, to do so, I need to scratch a little bit more on how the ellipsoid algorithm works to solve LPs.

Suppose we want to solve: $\max\{c^\top x : Ax \leq b\}$. First, we convert this optimization problem into a feasibility problem. We first guess the value of the LP, call the guess M. Then we try to check the feasibility of the following system: $\{Ax \leq b, c^\top x \geq M\}$. Suppose we did have a subroutine which either said NO to the feasibility question or said YES and returned a feasible solution, then we can perform a binary search on the value of M to in fact get the LP value and the optimal solution.

To give more details which we didn't cover in class, assume c's are integral. Let L, U be an upper bound on the value of OPT. Binary search gives us two values M_- and M_+ such that the system with $c^\top x \geq M_+$ is infeasible while the system with $c^\top x \geq M_-$ is feasible. So, $M_- \leq LP \leq M_+$. If $\delta = M_+ - M_-$, then the binary search needs to be run for $\log_2\left(\frac{U-L}{\delta}\right)$ iterations. Note that $M_- \geq LP - \delta$. How small does δ need to be so that we can assert that indeed $M_- = LP$? Well, if the feasibility subroutine always returned a feasible solution (when feasible) x which was a rational with both numerator and denominator *binary length* bounded by a known polynomial $\texttt{poly}(n)$ in the LP input size, then if we chose $\delta \le \frac{1}{2^{2\texttt{poly}(n)}}$, then in fact $M_- = LP$. This is because for any two $x \neq x'$ which are poly-bounded, $c^\top(x - x')$ would be at least $\frac{1}{2^{\texttt{poly}(n)}}$.

In sum, it suffices to solve the feasibility problem: $\{Ax \le b, c^\top x \ge M\}$ for any M. The **ellipsoid** algorithm needs the following subroutine called the separation oracle: given a candidate feasible solution x_t, either the oracle says feasible and we are done, or it returns a violated constraint. The ellipsoid algorithm uses this violated constraint to **deterministically** return another candidate solution x_{t+1}, and this process continues. The remarkable fact is that in number of steps polynomial in the input size, either we get a feasible solution guaranteed by the separation oracle, or the ellipsoid algorithm asserts that there is **no** feasible solution. This is precisely what was needed for the binary search algo, and we can use this in conjunction of the old idea to solve $\max\{c^\top x : Ax \le b\}$.

Now comes the crucial part. Consider the run of ellipsoid algorithm as above, and let T be the set of violated constraints returned by the separation oracle for all possible guesses. Consider the LP: $LP' = \max\{c^\top x : \hat{A}x \le \hat{b}\}$, where we only consider the violated constraints. What can we say about LP' and LP? Well, $LP' \ge LP$ surely. But we assert $LP' = LP$. This is because, the run of the ellipsoid algorithm is **the same** with LP and LP'! Why is this interesting? Well, say A has exponentially many constraints, and yet there was a separation oracle. Then the ellipsoid algorithm allows us to get a polynomial sized set of constraints which have the same LP value!

Let's come to what I promised – solution to LPs with large number of variables. This is what we do – we look at the dual. This has polynomially many variables but exponentially many constraints. Suppose we had a separation oracle for the dual. Then, as argued above, we can find polynomially many constraints in the dual such that the dual value doesn't change if we throw away the remaining constraints. Note constraints in dual correspond to variables in the primal. So, we can **throw** away all the variables in the primal except the ones corresponding to the constraints in the dual that the separation oracles asks us to keep, and the LP value doesn't change. Now we have a compact LP in the primal, which we can again solve to get the succinct primal solution.

4 Online Set Cover via Primal Dual + Randomized Rounding.

We now look at a different take on the set cover problem. We are given m sets, and for this lecture just assume the cost of each set is 1. However, we do not know the elements they cover! Rather, the elements arrive "online" in some adversarial order. When an element j arrives, we get to know which all sets contain it. The algorithm **must** pick one of these sets (unless, some set picked in the past already covers this element). After all the elements have arrived, we end up with a set cover $\mathcal{S}$. We would like to compare it with the optimum set cover of these elements, that is, we are comparing ourselves with an omniscient and all-powerful algorithm. Remarkably, we can get a decent approximation factor. For this lecture, we assume that we know n, the number of elements that will arrive.

The algorithm actually maintains two solutions (x, y) for the primal and the dual LP. Note that whenever an element arrives, it introduces a new constraint in the primal LP and a new variable in the dual LP. (To be precise, we should say that the solutions are (x^j, y^j) after the jth element arrives, but we delete the superscripts for brevity's sake). When element j arrives, the algorithm increases the x-value of the sets S_i that contain j and also increases the value y_j. This is done in such a way that the increase in the primal solution is comparable to the increase in the dual objective. Furthermore, we see in the end that the final dual (after all elements have arrived) is not too bad – we will see that $y/\log m$ is in fact dual feasible. This shows that the cost of the final primal solution, that is a **fractional** set cover, is at most $O(\log m)$ times the optimum.

However, we want an integral set cover. This is achieved by randomization. For each set S_i, we sample in the beginning certain thresholds θ_i. Each θ_i is chosen from $(0, \frac{1}{\ln n})$ uniformly at random. We pick a set S_i whenever x_i exceeds θ_i. Therefore, when an element j appears, we increase some of the x_i's. If one of them has exceeded θ_i, then we pick that set and we are done. Otherwise, we just pick an arbitrary set containing j.

The whole algorithm is described below.

1. Initially, set $x_i = 1/m$ for all sets S_1 to S_m. For each set S_i, select a **threshold** $\theta_i \in (0, \frac{1}{\ln n}]$ uniformly at random.
2. When element j arrives and we get to know the sets S_i with $j \in S_i$ do the following
   (a) Initialize $y_j = 1$.
   (b) While $\left(\sum_{i:j\in S_i} x_i < 1\right)$:
      i. Set $x_i = 2x_i$ for all i such that $j \in S_i$.
      ii. Set $y_j = y_j + 1$.
   (c) Pick all sets S_i with $x_i \geq \theta_i$. If no set containing j has been picked, then pick an arbitrary set S_i containing j.

The above algorithm is a feasible algorithm, in that, we always pick a set which contains the element new element j. To argue about the cost, one has three steps.

1. Prove that $\sum_{i=1}^{m} x_i \leq 1 + \sum_{j=1}^{n} y_j$.
2. Prove that $y/\log_2 m$ is dual feasible.
3. Prove that $\mathbf{Exp}[|\mathcal{S}|] \leq (\ln n) \cdot \sum_{i=1}^{m} x_i + 1$. From (1) and (2), we get that the RHS is at most $O(\log m \log n)$ times OPT.

The first part is easy. Consider the steps 2.b.i and 2.b.ii when x, y are modified. In each step $\sum_j y_j$ increases by exactly 1. $\sum_i x_i$ increases by $\sum_{i:j\in S_i} x_i$ which is < 1 since we run the while loop. Initially, $\sum_{i=1}^{m} x_i = 1$.

Now pick any set S and consider $\sum_{j\in S} y_j$. Whenever, any y_j increased by $+1$ in step 2.b.ii, we know that x_i doubled. Also note that x_i never can exceed 1. So, the number of time y_j can increase for some $j \in S$ is at most $\log m$. This proves $\sum_{j\in S} y_j \leq \log_2 m$.

Finally, let's argue about the expected number of sets in the solution. To that end, let's divide these sets into two – good sets, which were picked when $x_i \geq \theta_i$, and bad sets which an element j had to pick because no set covering it has $x_i \geq \theta_i$. Let's first argue about the number of bad sets picked. To that end, fix an element j and consider the probability that after the x has been modified after it has arrived, none of the sets containing it have $x_i \geq \theta_i$. Clearly at this time all $x_i \leq \frac{1}{\ln n}$. Now, consider θ_i being sampled right after x_i has been fixed – this causes no difference since the evolution of x_i didn't take θ_i into account. Therefore, the probability (over sampling of the threshold θ_i) that $x_i < \theta_i$ is exactly $(1 - \ln n x_i)$. Therefore, the probability that j is not covered by a good set is at most $1/n$, since $\sum_{i:j\in S_i} x_i \geq 1$. Therefore, the expected number of bad sets is ≤ 1.

Now, the probability that S_i is picked as a good set is precisely the probability $x_i \geq \theta_i$. If $x_i \geq \frac{1}{\ln n}$, this probability is 1 and is therefore at most $\ln n \cdot x_i$. Otherwise, it is precisely $(\ln n) \cdot x_i$. This completes the proof.