, 5

CMPUT 675: Approximation Algorithms **Fall 2013**

Lecture 4, 5 (Sep 17, Sep 19, 2013): Set Cover, LP Duality, 0-1 Knapsack

Lecturer: Mohammad R. Salavatipour *Scribe: Afshin Arefi, Based on older notes*

This week we see two other algorithms for approximating set cover: one using randomized rounding of an LP relaxation and the other using Primal-Dual schema. We will also see how to design an FPTAS for the 0/1 knapscak problem.

4.1 Set Cover using Randomized Rounding

In this section we consider another method for solving the set cover problem approximately. The method uses randomized rounding of the solution obtained from an LP relaxation of set cover. The basic idea behind this algorithm is first to solve the LP relaxation for the set cover problem. We can think of the solution $\vec{x}$ as a probability to either select a set or not. Note that the output of the algorithm might not be a set cover. But the probability to get a set cover can be increased by repeating the same procedure multiple times.

For set cover corresponding LP is

$$\begin{aligned} \text{minimize}: \quad & \sum_{i=1}^{m} c(S_i) x_{S_i} & & \\ \text{subject to}: \quad & \sum_{e \in S_i} c(S_i) x_{S_i} & \geq 1, & \quad \forall e \in U \\ & x_{S_i} & \geq 0, & \end{aligned}$$

Let α be a constant s.t. $e^{-\alpha \ln n} < 1/(4n)$.

Algorithm: Set cover by randomized rounding

1. Let x^* be an optimum solution to the LP relaxation for set cover
2. For each set S_i, set $\hat{x}_{S_i} = 1$ with probability $x^*_{S_i}$.
3. Let $C = \{S_i | \hat{x}_{S_i}\}$
4. Repeat the above procedure $\alpha \log n$ times and return the union of all sets C generated.

First we bound the expected cost of the set C computed in each iteration of step 3:

$$E\left[cost\left(C\right)\right] = \sum_{S_i} c(S_i) x^*_{s_i} = z^* \tag{4.1}$$

where z^* is the optimal value of the LP solution. But the problem with this set is that the solution might not be a set cover by itself. To address this, we repeat this procedure $\alpha \ln n$ times; we will show that with sufficiently high probability the returned solutionis a good set cover. That is, if C_i is the collection of sets that are selected at the i^{th} iteration of this algorithm, then we would have the final answer as $\mathcal{C} = \cup_{i=1}^{\alpha} c_i$.

Consider an arbitrary element e_j and WLOG assume that it belongs to sets $S_1, \ldots, S_k$. Since we start from a feasible solution, we must have $x^*_{S_1} + \ldots + x^*_{S_k} \geq 1$; in the worst case we should have $x^*_{S_1} + x^*_{S_2} + \ldots + x^*_{S_k} = 1$.

What is the probability that none of the corresponding k sets be selected given the probabilities of $x^*_{S_i}$'s? The following lemma (whose proof is starightforward) suggests that this probability is the highest when all the x^*'s are equal.

Lemma 1 *Given $x_1 + \ldots + x_k = 1$, if we select each S_i with probability x^*_i, the probability that no S_i will be selected is highest (worst case) when all $x^*_i = \frac{1}{k}$.*

Thus the probabilty that an element e_j is not covered in the i'th iteration of this algorithm is:

$$\Pr\left[e_j \notin C_i\right] \leq \left(1 - \frac{1}{k}\right)^k \leq e^{-1} \tag{4.2}$$

Now one way to increase the probability to get a set cover is to repeat the algorithm $\alpha \ln n$ times and take union of all the C's i.e. sets selected in each iteration; this is exactly what we do. Thus the probablity of an element not covered after $\alpha \ln n$ iteration is

$$\begin{aligned} \Pr\left(e_j \notin \bigcup_{i=1}^{\alpha \ln n} C_i\right) &\leq e^{-\alpha \ln n} && (4.3)\\ &= \frac{1}{4n} && (4.4)\end{aligned}$$

Using union bound, the probability that the result after $\alpha \ln n$ iterations is not a set cover is at most

$$\begin{aligned} \Pr\left(\bigcup_{i=1}^{\alpha} C_i \text{ is not a set cover}\right) &\leq \frac{n}{4n} && (4.5)\\ &= 1/4. && (4.6)\end{aligned}$$

The expected cost for the probablistic rounding set cover after $\alpha \ln n$ iterations is

$$\begin{aligned} E\left[cost\left(\bigcup_{i=1}^{\alpha} C_i\right)\right] &= \alpha \ln n \cdot z^* && (4.7)\\ &= O\left(\log n \cdot opt\right) && (4.8)\end{aligned}$$

Using Markov's inequality, $\Pr[X \geq t] \leq \frac{E[X]}{t}$:

$$\Pr\left[cost(C) > 4\alpha \cdot z^*\right] \leq \frac{1}{4}$$

Therefore, with a probability greater than or equal to $\frac{1}{2}$, we have a feasible solution with cost less than or equal to $4\alpha \cdot z^*$ which is in $O(\log n) \cdot OPT$, where OPT is the cost of the optimal solution. To get better probability it's enough to repeat the algorithm a constant number of times and get the best answer.

4.2 Duality of Linear Program

Consider the following simple LP:

$$\begin{aligned}
\text{minimize} \quad & 10x_1 + 6x_2 + 4x_3 && (4.9)\\
\text{subject to:} \quad & 2x_1 + x_2 - x_3 \geq 2 && (4.10)\\
& x_1 + x_2 + x_3 \geq 3 && (4.11)\\
& x_1, x_2, x_3 > 0 && (4.12)
\end{aligned}$$

Let z^* denote the optimum value of this linear program. Consider the question, "Is z^* at most 50?" A Yes certificate for this question, is a feasible solution with value at most 50, for example $x = (1, 1, 1)$, since it satisfies the two constraints of the problem, and the objective function value for this solution is 20. Thus, any Yes certificate to this question provides an upper bound on z^*.

Now consider the following question: is $z^* \geq 10$? To answer this we need to find lower bounds for all feasible solutions. We can obtain lower bounds by looking at the constraints. For instance, from constraint 4.10, and because the coefficients of variables x_1, x_2, x_3 are all smaller in the constraint with respect to those in the objective function, it follows that 2 is a lower bound for z^*. By combining constraints 4.10 and 4.11 we obtain that $3x_1 + 2x_2 + 0x_3 \geq 5$ and therefore 5 is a new lower bound for z^*. In general, any linear combination of these constraints could lead to a lower bound, as long as the final coefficients of the variables are not larger than those in the objective function. For instance, using a y_1 factor of constraint 1 and y_2 factor of constraint 4.11, we get:

$$\begin{aligned}
y_1(2x_1 + x_2 - x_3) &\geq 2y_1\\
y_2(x_1 + x_2 + x_3) &\geq 3y_2
\end{aligned}$$

There is a systematic way of obtaining the dual of any linear program; one is a minimization problem and the other is a maximization problem. In general, for a primal LP of the form:

$$\begin{array}{lrcl}
\text{minimize} & \sum_{i=1}^{n} c_i x_i & & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i\\
& x_j & \geq & 0
\end{array}$$

The dual has the form:

$$\begin{array}{lrcl}
\text{maximize} & \sum_{i=1}^{n} b_i y_i & & \\
\text{subject to} & \sum_{i=1}^{m} a_{ij} y_i & \leq & c_j\\
& y_i & \geq & 0
\end{array}$$

By construction, every feasible solution to the dual program gives a lower bound on the optimum value of the primal. Also, every feasible solution to the primal program gives an upper bound on the optimal value of the dual. Therefore, if we can find feasible solutions for the dual and the primal with matching objective function value, then both solutions must be optimal.

Theorem 1 ***Weak Duality Theorem:*** *If $\vec{x}$ and $\vec{y}$ are feasible solution for primal and dual then $\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$.*

Proof.

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n}\left(\sum_{i=1}^{m} a_{ij} y_i\right) x_j \quad (4.13)$$

$$= \sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij} x_j\right) y_i \quad (4.14)$$

$$\geq \sum_{i=1}^{n} b_i y_i \quad (4.15)$$

■

Theorem 2 (Strong Duality Theorem) *Primal of an LP has a finite optimum solution if and only if its dual has finte optimum. Also if $\vec{x^*}$ and $\vec{y^*}$ are optimal for primal and dual then $\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{n} b_i y_i$*

The following theorem follows easily from the above theorems:

Theorem 3 (Complementary slackness condition) *Let $\vec{x}$ and $\vec{y}$ be two feasible solutions for the primal and dual. then $\vec{x}$ and $\vec{y}$ are both optimum iff the following two conditions hold:*

1. ***Primal complementary slackness conditions***
   For each $1 \leq j \leq n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$;

2. ***Dual complementary slackness conditions***
   For each $1 \leq i \leq m$: either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

Relaxed complementary slackness condition: Let $\vec{x}$ and $\vec{y}$ be the feasible solution for primal and dual. Suppose that we have both primal and dual *relaxed* slackness conditions: for $\alpha \geq 1$ and $\beta \geq 1$, if

- for each $1 \leq j \leq n$ either $x_j = 0$ or $\frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$; and
- for each $1 \leq i \leq m$ either $y_i = 0$ or $b_i \leq \sum_{j=1}^{m} a_{ij} x_j \leq \beta b_j$,

then $\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \sum_{i=1}^{m} b_i y_i$.

4.3 Primal dual scheme

Previously mentioned LP based algorithms actually approximate the problem by solving the LP and computing an integer solution from the LP solution. The general idea of primary-dual method is to start with a primal infeasible and a dual feasible solution (usually the trivial solution $\vec{x} = 0$ and $\vec{y} = 0$). Then we iteratively improve the feasibility of primal and optimality of dual. Primal is always extended integrally and at the end Primal is a feasible solution. At each iteration, we use relaxed slackness conditions to help to find feasible solutions to the primal.

Big advantage of Primal-Dual over rounding: we don't have to solve LP (which is time consuming although polynomial time solvable).

4.3.1 primal dual applied to set cover

Consider the following LP relaxation for Set Cover and its dual:

$$
\begin{array}{rrcl}
\text{minimize} & \sum_{S\in\mathcal{S}} c(S)x_S & & \\
\text{subject to} & \sum_{S:e\in S} x_S & \geq & 1, \quad \forall e\in U \\
 & x_S & \geq & 0
\end{array}
\qquad
\begin{array}{rrcl}
\text{maximize} & \sum_{e\in U} y_e & & \\
\text{subject to} & \sum_{S:e\in S} y_e & \leq & c(S) \quad \forall S\in\mathcal{S} \\
 & y_e & \geq & 0
\end{array}
$$

Note that the dual problem is a packing problem. We can say, we are going to pack stuff into elements so that the total amount packed is maximized without overpacking any set.

Remark: The dual of a covering problem is a packing problem and the dual of a packing problem is a covering problem. Packing problems are typically harder.

The frequency of each element $e \in U$ in the set cover problem is the number of sets that contain that element e. We use f to denote the maximum frequency. Our goal is to design a primal-dual f-approximation for set cover.

We start with a trivial solution of $\overrightarrow{x} = 0$, $\overrightarrow{y} = 0$. We try to satisfy the Primal complementary slackness conditions:

$$\forall S \in \S : x_S \neq 0 \Rightarrow \sum_{e\in S} y_e = c(S).$$

If we find a pair of feasible solutions (x, y) that satisfy the above condition, because each $x_s > 0$ has value 1, we'll have:

$$x_s = 1 \Rightarrow \sum_{e\in S} y_e = c(S).$$

In that case, the total cost of the solution will be:

$$
\begin{aligned}
\sum_{S\in\mathcal{S}} c(S)\cdot x_S = \sum_{S\in\mathcal{S}} x_S \left(\sum_{e\in S} y_e\right) & \leq \sum_{e\in U} y_e \sum_{S:e\in S} \\
& \leq \sum_{e\in U} y_e \cdot f \\
& \leq f \cdot \mathrm{opt}_{LP}
\end{aligned}
$$

So it is enough to show how find such feasible primal solution. We call a set S for which the right-hand-side inequality of the primal complementary slackness condition holds with equality a tight set (intuitively we cannot pack more stuff into elements of that set).

The following is the **PrimalDual Set Cover Algorithm**.

$\vec{x} \leftarrow 0$, $\vec{y} \leftarrow 0$
while not all elements are covered **do**
  pick an uncovered element e, raise y_e until some set goes tight
  pick all tight sets and update $\vec{x}$
  declare all the elements in those sets as covered
endwhile
output the set cover $\vec{x}$

Theorem 4 *This algorithm is an f-approximation for set cover.*

Proof. Consider primal complementary slackness

$$\text{if } x_{S_i} > 0 \text{ then } \sum_{e \in S_i} y_e = C(S_i) \tag{4.16}$$

No if we find a easible solution to primal LP $\vec{x}$ and dual LP $\vec{y}$ ensuring that $\vec{x}$ is integer (i.e each $x_{S_i} > 0$ is 1) then

$$\text{if } x_{S_i} = 0 \text{ then } \sum_{e \in S_i} y_e = C(S_i) \tag{4.17}$$

and total cost will be

$$\begin{aligned}
\sum_{S_i} c_{S_i} \cdot x_{S_i} &= \sum_{S_i} x_{S_i} \left(\sum_{e \in S_i} y_e \right) && (4.18)\\
&\leq \sum_{e \in S_i} y_e \sum_{S_i : e \in S_i} x_{S_i} && (4.19)\\
&\leq \sum_{e \in S_i} y_e \cdot f \text{ where f is frequency of element} && (4.20)\\
&\leq f \cdot opt_L P && (4.21)
\end{aligned}$$

■

4.4 0-1 Knapsack

The 0-1 knapsack problem can be described as, given a set of items, each having value v_i and weight w_i and a knapsack of capacity B, find a subset of items that fit into the knapsack while maximizing their total value, i.e.:

Input: Set of items $\{1, ..., n\}$, each item has value v_i and a weight w_i, we have a knapsack with capacity B, $v_i, w_i, B \in Z^+$.
Goal: Find a subset S of items such that $\sum_{i \in S} v_i$ is maximum and $\sum_{i \in S} w_i \leq B$.

Here we present an FPTAS for knapsack. Clearly, if the weight of a single item is larger than B we can ignore that item. So let's assume $\forall i : w_i \leq B$.

The obvious greedy algorithm is to sort in non-increasing order of $\frac{v_i}{w_i}$ and pick the objects in this order. Unfortunately this greedy algorithm has a bad solution. In fact, knapsack is an NP-hard optimization problem. First we present a dynamic Programming algorithm for knapsack. Let V be the largest value among all items and OPT be the value of an optimum solution. Clearly OPT $\leq nV$. We have an $n \times (nV+1)$ table A. Let $A[i, v]$ be the smallest weight of a subset of items from $\{1, \ldots, i\}$ such that the value of the items is exactly equal to v, if no such set exists then $A[i, v] = \infty$. Then $A[1, v]$ is easy to compute for any value $v \in \{1, \ldots, nV\}$ and

$$A[i, v] = \begin{cases} \min\{A[i-1, v], A[i-1, v - v_i] + w_i\} & \text{if } v_i < v \\ A[i-1, v] & O.W. \end{cases}$$

This leads to the following algorithm:

The running time of this algorithm is $O(n^2 \cdot V)$. But we know that knapsack is NP-hard. So have we proved that P=NP?! This is NOT polynomial in the size of input because V is not polynomial in size of v_i's (represented in binary). We only need $\log V$ bits to represent V, so $n^2 V$ is exponential in V. This is polynomial only if the input is given in unary representation. For this reason, we call this algorithm a *pseudo-polynomial* time algorithm.

Dynamic Programming algorithm for Knapsack

$V = \max_{1 \le i \le n} V_i$
for $i = 1$ to n do $A[i, 0] = 0$
for $v = 1$ to nV do
$$A[1, v] = \begin{cases} w_1 & if \quad v_1 = v \\ \infty & \texttt{otherwise} \end{cases}$$
for $i = 2$ to n do
    for $v = 1$ to nV do
        if $v_i \le v$ then
            $A[i, v] = \min\{A[i-1, v], A[i-1, v-v_i] + w_i\}$
        else
            $A[i, v] = A[i-1, v]$

Figure 4.1: Dynamic Programming Algorithm for Knapsack.

The main reason that Dynamic Programming is not polynomial time is that the values of items can be much larger than n. If they were all polynomially bounded by n, then this algorithm would be polynomial time. We are going to use this fact to design an FPTAS for knapsack. To do so, we are going to use only a polynomially bounded segments of values that will depend on n and $\frac{1}{\varepsilon}$ (the error parameter). Then we will find a solution (in polynomial time) that is at least $(1 - \varepsilon)$.OPT using dynamic programming.

4.4.1 FPTAS algorithm for 0-1 Knapsack

FPTAS algorithm for knapsack

Let $k = \frac{\epsilon V}{n}$.
for each item i **let** $v_i^{'} = \lfloor \frac{v_i}{k} \lfloor$
Run the above dynamic program with same weights w_i but with value $v_i^{'}$.
Let $S^{'}$ be the solution of dynamic program, return $S^{'}$

Theorem 5 *The above algorithm is an FPTAS for knapsack.*

Proof. First note that the largest value V' in the new instance is

$$V^{'} = \left\lfloor \frac{V}{\epsilon V/n} \right\rfloor \tag{4.22}$$
$$= \frac{n}{\epsilon} \tag{4.23}$$

Thus the resulting time complexity of the algorithm is $O\left(\frac{n^3}{\epsilon}\right)$ which is polynomial in both n and ϵ. Now we prove the approximation ratio.

Let S be an optimum solution and OPT be the value of S. For each item i: $kv_i' \le v_i \le k(v_i' + 1)$. Therefore:

$$v_i - k \le kv_i' \tag{4.24}$$

and

$$v_i \ge kv_i' \tag{4.25}$$

So by (4.24): OPT $= \sum_{i \in S} v_i \leq k. \sum_{i \in S} v_i' + kn$.

and we know that value of $S^{'}$ is the best under $v_i^{'}$. Now

$$\begin{aligned}
\sum_{i \in S'} v_i &\geq k \sum_{i \in S'} v_i^{'} && (4.26)\\
&\geq k \sum_{i \in S} v_i^{'} && (4.27)\\
&\geq \sum_{i \in S'} v_i - nk && (4.28)\\
&= opt - \epsilon V && (4.29)\\
&\geq opt\,(1 - \epsilon)\,, && (4.30)
\end{aligned}$$

since $V \leq$ OPT and therefoer $\epsilon V \leq \epsilon$OPT. ■

4.4.2 Pseudo polynomial time algorithms and strong NP hard problems

An algorithm has pseudo-polynomial time algorithm if its running time is polynomial when the input is represented in Unary. A problem is strongly NP-hard if it has no pseudo-polynomial time algorithm unless P=NP. The following theorem establishes a connection between having an FTPAS and having a pseudo-polynomial time exact algorithm.

Theorem 6 *Let $|I|$ be the size of input in binary, $|I_u|$ be the size of instance I of problem Π represented in unary. Suppose that I is an NP hard minimization problem s.t. the objective function is integer for any instance I and $opt\,(I) \leq P\,(|I_u|)$ where P is some poly function. Then if Π has an FPTAS then it is not strongly NP hard and has a pseudo polynomial time algorithm.*

Proof. Let A be and FPTAS for problem Π with time $Q\,(|I|\,, \epsilon)$ which is polynomial time in both input size and ϵ. Let $\epsilon = \frac{1}{P(|I_u|)}$. Now the as it is assumed A is FPTAS thus

$$\begin{aligned}
\text{solution} &\leq (1 + \epsilon)\, opt && (4.31)\\
&= \left(1 + \frac{1}{P\,(|I_u|)}\right) opt && (4.32)\\
&= \text{opt} + \frac{\text{opt}}{P\,(|I_u|)} && (4.33)\\
&< 1 + opt && (4.34)
\end{aligned}$$

■