CMPUT 675: Topics on Approximation Algorithms and Approximability **Fall 2015**

Lecture 4: September 10, 2015

Lecturer: Mohammad R. Salavatipour *Scribe: Nadia Ady*

4.1 Recall: Set Cover via Randomized Rounding

In our last lecture, we introduced an approximation algorithm for Set Cover using Randomized Rounding.

4.1.1 Randomized Rounding

SC-Rand-Rounding

```
1  let x* be an optimum LP solution for Set Cover
2  C_i ← ∅ for 1 ≤ i ≤ α log n
3  for i ← 1 to α log n
4          do for each s ∈ S
5                  do add s to C_i with probability x*_s
6  return ⋃ C_i
```

For one round of the outer for-loop in lines 3-5,

$$E[\text{cost}(C)] = \sum_{i=1}^{n} c(S_i) \cdot \Pr[S_i \text{ is chosen}] \tag{4.1}$$

$$= \sum_{i=1}^{n} x^*_{S_i} \cdot c(S_i) \tag{4.2}$$

$$= \text{OPT}_{\text{LP}} \tag{4.3}$$

This implies that the total cost $\sim O(\log n \cdot \text{OPT}_{\text{LP}})$.

Lemma 4.1 *If we choose α large enough that $e^{\alpha \log n} \leq \frac{1}{4n}$, then the probability that there is some uncovered element is at most $\frac{1}{4}$.*

Proof: Let α be large enough that $e^{\alpha \log n} \leq \frac{1}{4n}$. Consider an arbitrary element e_j and suppose it belongs to k sets $S_1, ..., S_k$. Since we are starting with a feasible solution, we have $x^*_1 + x^*_2 + ... + x^*_k \geq 1$.

The probability that e_j is covered in a single iteration of the loop is:

$$\Pr[e_j \text{ is covered}] = 1 - \prod_{l=1}^{k} \left(1 - x^*_{S_l}\right) \tag{4.4}$$

It is a straightforward exercise to show that the worst case occurs (ie. the probability that no $s \in S$ will be selected in the iteration is highest) when $x^*_{S_l} = \frac{1}{k}$ for $1 \leq l \leq k$. In this case,

$$\Pr[e_j \notin C_i] \leq \left(1 - \frac{1}{k}\right)^k \leq e^{-1} \tag{4.5}$$

The probability that e_j is not covered after $\alpha \log n$ iterations is:

$$\Pr[e_j \text{ is not covered at the end}] \leq e^{-\alpha \log n} \leq \frac{1}{4n} \tag{4.6}$$

$$\implies \Pr[\text{there is some uncovered element}] \leq \frac{1}{4} \tag{4.7}$$

■

Lemma 4.2 *The probability that the cost of the collection C is at least $4 \log n \cdot OPT_{LP}$ is less than $\frac{1}{4}$.*

Proof: We recall Markov's inequality.

$$\Pr[x > t] \leq \frac{E[x]}{t}$$

From the equality in equation 4.1, this implies that

$$\Pr[\text{cost}(C) > 4 \log n \cdot \text{OPT}_{\text{LP}}] \leq \frac{1}{4} \tag{4.8}$$

■

By combining the results of the preceding two lemmas we can see that, with probability at least $\frac{1}{2}$, we will have a solution where each e_j is covered by an element in C (the solution is feasible) and the total cost is at most $O(\log n \cdot \text{OPT}_{\text{LP}})$. To increase this probability, it is sufficient to increase the number of iterations.

4.2 Polynomial-time Approximation Schemes (PTAS)

For any fixed $\varepsilon > 0$, a PTAS provides a $(1 + \varepsilon)$-approximation with time polynomial in n.
Similarly, for any fixed $\varepsilon > 0$, an FPTAS provides a $(1 + \varepsilon)$-approximation with time polynomial in n and $\frac{1}{\varepsilon}$.

4.2.1 Knapsack

In the Knapsack problem, our input is a collection of n items and a capacity. Item i has value $v_i \in \mathbb{Z}^+$ and weight $w_i \in \mathbb{Z}^+$. Our knapsack has a capacity of $B \in \mathbb{Z}^+$. The optimization problem is to select a subset of items which maximize the total value $\sum_{i=1}^n v_i$ subject to the constraint that the total weight must be at most B (i.e. $\sum_{i=1}^n w_i \leq B$).

4.2.1.1 Natural Greedy Knapsack

The natural greedy algorithm is simply to sort the items by decreasing $\frac{v_i}{w_i}$ and pick the items in that order. This algorithm is a 2-approximation.

Example 4.3

$$B = 20$$
$$v_1 = 10, w_1 = 10$$
$$v_2 = 10, w_2 = 10$$
$$v_3 = 12, w_3 = 11$$

Consider what happens when you multiply all of these values by $\frac{1}{\varepsilon}$ for ε arbitrarily close to zero.

4.2.1.2 Dynamic Programming Knapsack

Say $\max_{1\leq i\leq n} v_i = V$ and assume that $w_i \leq B$ for all $1 \leq i \leq n$. Let us define for $1 \leq i \leq n$ and $0 \leq v \leq n \cdot V$:

$$A[i,v] = \begin{cases} \text{the min weight of a packing using items } 1,...,i \text{ with total value } v, \textit{or} \\ \infty \text{ if there is no such solution} \end{cases} \tag{4.9}$$

Our aim is to find the max v such that $A[n,v] \leq B$.
Observe that, for each i, we either use item i or we don't, so we can define $A[v,i]$ recursively:

$$A[i,v] = \min \begin{cases} A[i-1,v], \\ A[i-1,v-v_i] + w_i \end{cases} \tag{4.10}$$

DymProg-Knapsack

for $i \leftarrow 1$ to n
    do $A[i,0] \leftarrow 0$
for $v \leftarrow 1$ to $n \cdot V$
    do if $v = v_1$
        then $A[1,v] \leftarrow w_1$
        else ∞
      for $i \leftarrow 2$ to n
          do for $v \leftarrow 1$ to $n \cdot V$
                do $A[i,v] \leftarrow \min\{A[i-1,v], A[i-1,v-v_i]+w_i\}$

The running time of this algorithm is $O(n^2V)$. However, this is not polynomial in the size of the input because V is not polynomial in size of the input. We need $\log V$ bits to represent V. We call this a *pseudopolynomial*-time algorithm. However, this will lead us to an FPTAS for Knapsack:

1. Let $k = \frac{\varepsilon V}{n}$ and for $1 \leq i \leq n$, let $v'_i = \left\lfloor \frac{v_i}{k} \right\rfloor$
2. Run DymProg-Knapsack using input items $1,...,n$ with each item i having weight w_i but value v'_i.
3. Let S' be the solution returned.
4. Return S.

Theorem 4.4 *This is an FPTAS for Knapsack.*

Proof: Suppose S is an optimum solution and has value OPT.
Observe that for $1 \leq i \leq n$,

$$kv'_i \leq v_i \leq k(v'_i + 1) \tag{4.11}$$

$$\implies \text{OPT} = \sum_{i\in S} v_i \leq k \sum_{i \in S} v'_i + kn \tag{4.12}$$

Notice that the value of S' is optimum for v'_i values.

$$\text{our solution} = \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \tag{4.13}$$

This implies that:

$$\sum_{i \in S'} v_i \geq \sum_{i \in S'} k v_i' \tag{4.14}$$
$$\geq k \sum_{i \in S} v_i' \tag{4.15}$$
$$\geq \text{OPT} - nk \tag{4.16}$$
$$\geq \text{OPT} - \varepsilon V \tag{4.17}$$
$$\geq (1-\varepsilon)\text{OPT} \tag{4.18}$$

■

Most NP-complete problems are strongly NP-hard; that is, they don't have pseudo-polytime algorithms.

Theorem 4.5 *Suppose that π is an NP-hard minimization problem such that the objective function is always integer on any instance I of π and $OPT(I) < p(|I_u|)$ where p is some polynomial and $|I_u|$ is the size of I represented in unary. Then if π has an FPTAS then it is not strongly NP-hard.*

Sketch of Proof: Let $\varepsilon < \frac{1}{p(|I_u|)}$. Then the solution by an FPTAS has value at most

$$(1+\varepsilon)\text{OPT}(I) < \text{OPT}(I) + \frac{\text{OPT}(I)}{p(|I_u|)} \tag{4.19}$$
$$< \text{OPT}(I) \text{ and is an integer} \tag{4.20}$$
$$\implies = \text{OPT}(I) \tag{4.21}$$

4.2.2 Bin-Packing

The one-dimensional bin-packing problem is as follows: Given an input set of items $1, .., n$ with each item i having a size $s_i \in (0,1] \cap \mathbb{Q}^+$, the goal is to pack the items into as few unit-sized bins as possible.

Theorem 4.6 *There is no α-approximation for the bin-packing problem for any $\alpha < \frac{3}{2}$ unless $P = NP$.*

Proof: Consider the Partition problem (which is NP-hard.)
Given set $S = \{S_1, ..., S_n\} \subseteq \mathbb{Z}^+$, can we partition S into 2 sets A and B such that $\sum_{S_i \in A} S_i = \sum_{S_j \in B} S_j$?

Consider an instance I of the Partition problem normalized such that $\sum_{S_i \in S} S_i = 2$. We can, in polynomial time, convert instance I into an instance I' of bin packing such that S_i is the size of item i for $1 \leq i \leq n$. If all of the items from I' fit into 2 bins, then I' is a YES instance of Partition, otherwise it is a NO instance.

On the other hand, if we have a YES instance of Partition, I, then the corresponding instance of bin packing has a solution using two bins following the rule that if S_i is in A, it belongs in the first bin, and it belongs in the second bin otherwise. Since A and B are of equal size (that is ($\frac{2}{2} = 1$), we know that this is a valid bin packing.

Notice that since $\sum_{S_i \in S} S_i = 2$, an optimum bin packing for I' requires at least two bins (i.e. $\text{OPT}(I') \geq 2$). If we had an α-approximation for some $\alpha < \frac{3}{2}$, we could compute the cost of OPT, allowing us to solve the Partition problem on I exactly, which cannot occur unless $P = NP$. ■

4.2.2.1 First Fit (Greedy) Algorithm

FF-Bin-Packing

```
for i ← 1 to n
        do let j be the first bin into which you can fit item i.
            put item i into bin j
return the used bins
```

Theorem 4.7 *The cost of a first-fit solution is at most* $2 \cdot OPT + 1$.

Proof: Observe that the number of bins containing objects whose size sums to $\leq \frac{1}{2}$ is at most 1. This can be seen by noting that if you have $i < j$ such that both bin i and bin j are at most half full at termination of the algorithm, then at the time items were put into bin j, there was enough room for them to fit in bin i, contradicting our assumption that we followed the "first fit" policy. Therefore, if FF is the number of bins used by (i.e. the cost of) a first-fit solution, then $\frac{1}{2}(\mathrm{FF} - 1) \leq \sum_{i=1}^{n} s_i$.

It is clear that $\mathrm{OPT} \geq \sum_{i=1}^{n} s_i$.
It follows that $\mathrm{FF} \leq 2\mathrm{OPT} + 1$. ■

References

N3-13 M.R. Salavatipour (scribe: A. Arefi), Lecture 4, 5 (Sep 17, Sep 19, 2013): Set Cover, LP Duality, 0-1 Knapsack, *University of Alberta CMPUT 675: Approximation Algorithms Course Notes*, Fall 2013. Retrieved from http://webdocs.cs.ualberta.ca/∼mreza/courses/Approx13/week3.pdf

N4-13 M.R. Salavatipour (scribe: R. Sivakumar), Lecture 6,7 (Sep 24 and 26, 2013): Bin Packing, Facility Location, K-Center, *University of Alberta CMPUT 675: Approximation Algorithms Course Notes*, Fall 2013. Retrieved from http://webdocs.cs.ualberta.ca/∼mreza/courses/Approx13/week4.pdf