CS 5150/6150: Assignment 5+6

Wei Liu (u0614581, weiliu@sci.utah.edu)

December 17, 2010

1 Question 1

The 2-approximation algorithm: The algorithm looks like this:

Algorithm 1 2-approximation algorithm for E-commerce problem

```
    STATE Give $\{v_1, \dots, v_n\}$ and $\{A_1, \dots, A_m\}$ as input.while a new customer comes in do  show him the ads $\tilde{j}$
    with lowest total weight $\widetilde{V_j}$ among all ads up to now.
  end while
  return  The ads assignment and the lowest spread $s$.
```

2-approximation proof: First I try to find a upper bound of s^*, the spread value of the optimal solution. Just like in load-balancing problem, we do not know the exact solution and its spread value, but we can construct a upperbound and compare our approximated solution with that upperbound. Along this line, we see

$$s^* \le \frac{\overline{v}}{m} \tag{1}$$

Suppose $s^* > \frac{\overline{v}}{m}$, then all other ads have weights greater than $\frac{\overline{v}}{m}$, so the sum of weights $\overline{v} = \sum V_j < m \cdot \frac{\overline{v}}{m} = \overline{v}$, which is a contradiction.

With eq. (1) established, a good guess is we can prove for spread s in our approximated solution,

$$s > \overline{v}/(2m) \tag{2}$$

With eq. (1) and eq. (2) together, we can prove $s > (1/2)s^*$ with no problem. Now I will prove eq. (2). Let j_s be the ads that achieves the spread s, and let j be any ads other than j_s. Assume v_j is the last customer that comes to ads j. Just the moments before v_i comes in, ads V_j, as the total weights on j, must be less than spread s, so $V_j - v_j < s$, and $V_j < v_j + s < \overline{v}/(2m) + s$.

$$\sum_{k=1}^{m} V_k = s + \sum_{j \neq j_s}^{V_j} < s + (m-1)(\frac{\overline{v}}{2m} + s) = m \cdot s + \frac{m-1}{m}\overline{v}$$

If we suppose (2) is not true, then $s \le \overline{v}/(2m)$, we have

$$\sum_{k=1}^{m} V_k < \frac{\overline{v}}{2m} \cdot m + \frac{m-1}{m}\overline{v} = \frac{2m-1}{2m}\overline{v} \tag{3}$$

The left hand side of (3) is just the sum of all the weights, and (3) is a contradiction. hence we prove (2).

2 Question 2

Suppose there are N items and each has value v_n and weight w_n. The maximal weight is W. To define a integer programming problem, we define a indicator variable x_n for each item. $x_n \in \{0,1\}$, and 1 means x_n is put into the knapsack, while 0 mean it is not. The integer programming will look like:

$$\begin{aligned} \text{minimize} \quad & \sum_{n=1}^{N} v_n x_n \\ \text{subject to} \quad & \sum_{n=1}^{N} w_n x_n \leq W \\ & x_n \in \{0,1\} \text{ for all } n \in \{1,\dots,N\} \end{aligned}$$

To relax the IP problem to a LP problem, we need to relax the constraint of x_n and let $x_n \in [0,1]$, i.e. x_n takes any real value between 0 and 1. Because the integer solution is the subset of the linear programming problem's solution, the optimal value of LP is no less than the IP problem.

To solve this simple LP problem, we can sort all the item in non-increasing order of v_i/w_i. We take item in this order until the total weights exceed the allowed weight, i.e. $\sum_{n=1}^{K} w_n \leq W$. To make the total sum of weights of selected items equal to W, there will be one item whose x_k is a fractional value.

rounding and approximation: Now we need to round the LP to get an approximated solution of the original problem. How good the approximation is depend on the rounding scheme.

Rounding scheme one: We just discard the fractional item x_k and choose $x_1, \dots, x_{k-1}$ as the final solution. If $w_1 = 1, v_1 = 2$, and $w_2 = W, v_2 = W$, by this rounding scheme we would select the first item, since it has larger v/w ratio. However the second item is a better solution in the case of $W > 2$. This says the rounding scheme can be arbitrary bad.

Rounding scheme two: We choose one item with largest w_i and still less than W. We compare this solution and solution of round scheme one, and choose the better one.

2-Approximation: I claim rounding scheme two is a 2-Approximation. To prove this, we first see $\sum_{n=1}^{K} x_n \geq$ OPT, where OPT is the optimal solution of original KNAPSACK problem. Then either $\sum_{n=1}^{k-1}$ or x_k is greater than OPT/2 (otherwise the sum of them $\sum_{n=1}^{k} = \sum_{n=1}^{k-1} + x_k$ would be less than OPT). So if we choose a better solution between $\sum_{n=1}^{k-1}$ and x_k, the solution is at least OPT/2.

3 Question 3

Heuristics one: In each step we choose a v_i with largest weight w_i, and remove the possible neighbors v_{i-1} and v_{i+1}. So we have

$$\begin{aligned} w_i &\geq w_{i-1} \\ w_i &\geq w_{i+1} \\ 2w_i &\geq w_{i-1} + w_{i+1} \end{aligned}$$

We can compare this solution and optimal solution. We see this solution is less optimal only because in some steps, it choose some v_i not in optimal set, and discard its neighbors which may be in the optimal set. However, each time this happens, we choose v_i which has weights greater than half the weight of discarded nodes. In the worse case, all the nodes in optimal set is discarded and replaced with nodes not in the optimal set. In this case we still guarantee

$$W = \sum_{i}^{w_i} \geq \frac{1}{2}\left(\sum w_{i-1} + w_{i+1}\right) = \frac{1}{2}\text{OPT}$$

And we prove the 2-approximation.

Heuristics two: Suppose it is not 2-approximation. That means both the sets $\{v_, v_3, \dots\}$ and $\{v_2, v_4, \dots\}$ are less than $(1/2)W^*$, with W^* the optimal solution. Because $W^* < \sum_{i=1}^n w_i$, that means $(v_1 + v_3 + \dots) + (v_2 + v_4 + \dots) = \sum_{i=1}^n v_i < W^* < \sum_{i=1}^n w_i$, which is a contradiction. Therefore this heuristics is a 2-approximation.

4 Question 4

Following the hint, I will first build a Dynamic Programming (DP) algorithm which find the exact optimal solution. Then I will use scaling and rounding to get an approximate solution, just like KNAPSACK.

DP Solution: In DP solution of KNAPSACK, we draw a $N \times W$ table, with N is number of the items, and W is the total weights allowed. Here, because the objective function is a ratio of subset sum, we need to use $M = \sum_{i=1}^n a_i$ and draw two $n \times M$ table **A** and **B**. $\mathbf{A}(i,j) = 1$ when there is a subset $\tilde{S} = \sum_{k=1}^K a_k = j$ and i is the largest element in this subset. Also let $\mathbf{B}(i,j) = \tilde{S}$ if there is such subset. And the algorithm to fill table **A** and **B** is[1]

Algorithm 2 DP solution for Subset-Ratio problem

```
Let $\mathbf{A}(0,0) = 1$ and $\mathbf{B}(0,0) = \phi$
for $i = 1$ to $n$ do
    $\mathbf{A}(i,0) = 0$, $\mathbf{B}(i,0) = \phi$.
end for
for $j = 1$ to $M$ do
    $\mathbf{A}(0,j) = 0$, $\mathbf{B}(0,j) = \phi$.
end for
for $j = 1$ to $M$ do
    for $i = 1$ to $n$ do
        if $j \geq a_i$, and there exist $k < i$ such that $\mathbf{A}(k, j - a_i) = 1$ then
            $\mathbf{A}(i,j) = 1$, $\mathbf{B}(i,j) = B(i,j) \cup \{i\}$
        else
            $\mathbf{A}(i,j) = 0$, $\mathbf{B}(i,j) = \phi$.
        end if
        if $\mathbf{A}(i_1,j) = \mathbf{A}(i_2,j) = 1$ and $i_1 \neq i_2$ then
            STOP
        end if
    end for
end for
```

Analysis of the algorithm 2: Table **A** tell us which j is the sum of some elements. j is called a candidate if $\mathbf{A}(i,j) = 1$. For each such candidate, we look for any greatest candidate k, such that $k < j$ and $\mathbf{B}(i,j) \cap \mathbf{B}(i',k) = \phi$, meaning disjoint set. When we find all such k, we look for the k^* that has minimal $\frac{j}{k}$. And $\mathbf{B}(i,j)$ and $\mathbf{B}(i',k^*)$ is the subsets of the best solution.

For each $j \in \{1, M\}$, we need look at $k \times n$ number of $\mathbf{B}(i',k)$. So the computation cost of searching table **B** is $\mathcal{O}(nM^2)$. Together withe the cost of building table **A** and **B** is $\mathcal{O}(nM)$, the total time spent on this algorithm is $\mathcal{O}(nM^2)$, which is a pseudo-polynomial algorithm. When M is small, it can be seen as polynomial.

Scaling and Rounding: When the exact solution is available, we will use a scaling parameter b, and set the value of a_i to a multiple of b, i.e. $\tilde{a}_i = \lceil a_i/b \rceil b$. By doing this, the $M = \sum_{i=1}^n a_i$ will also be a multiple of b. Then, instead of solving the $\tilde{a}_i$ problem, we can solve $\hat{a}_i = \tilde{a}_i/b$ problem.

Proof of FPTAS: If we choose b as a function of ϵ like KNAPSACK-APPROX algorithm on the textbook, we can get a Full Polynomial Time Approximation scheme(FPTAS), and have

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq (1+\epsilon) \frac{\sum_{i \in S_1^*} a_i}{\sum_{i \in S_2^*} a_i}$$

That is, the approximated solution is with ϵ of optimal solution. And the approximation algorithm runs in polynomial time.

5 Question 5

A naive solution is to increase n until the approximated value reach close to e.

Algorithm 3 Find n that approximate e.

```
n = 1
while $(1+1/n)^n < (1-\varepsilon)e$ do
    $n := n+1$
end while
```

We use $1-\varepsilon$ because $(1+1/n)^n$ is a increasing function of n.

Algorithm 4 Find n that approximate $1/e$.

```
n = 1
while $(1-1/n)^n < (1-\varepsilon)(1/e)$ do
    $n := n+1$
end while
```

In above algorithm I assume e is know as a optimal solution. A better assumption is not to assume we know the optimal solution but establish a upper/lower bound of it. The final result for (a) is 50 and 500017. For (b) it is 50 and 500008.

6 Question 6

Upper Bound: We use Chebyshev's inequality (i.e. Markov inequality) to find the upper bound. Chebyshev's inequality says for any random variable X and $a > 0$

$$\Pr\{|X - \mathbb{E}(X)| \geq a\} \leq \frac{Var(X)}{a^2}.$$

In our case we can compute expectation and variance of X as

$$\mathbb{E}(X) = \mathbb{E}\left[\sum x_i\right] = \sum \mathbb{E}[X_i] = 100 \cdot 3.5 = 350$$

$$Var(X) = \sum \left(\mathbb{E}[X^2] - (\mathbb{E}[X])^2\right) = 290$$

If we let $a = 50$, we have

$$\Pr\{|X - 350| \geq 50\} \leq \frac{290}{50^2} = 0.116.$$

7 Question 7

We use Hoeffding's inequality to provide an upper bound on the probability of the sum of random variables. First we simplify the original inequality as

$$\begin{aligned}\Pr\{X_H - XT > c\sqrt{n}\} &= \Pr\{X_H - (2n - X_H) < c\sqrt{n}\} \\ &= \Pr\{2X_H - 2n > c\sqrt{n}\} \\ &= \Pr\{X_H - n > \frac{c}{2}\sqrt{n}\}\end{aligned}$$

X_H can be seen as sum of independent variable $X_H = \sum_{i=1}^{2n} X_i$, where X_i is indicator variables. $X_i = 1$ means head and $X_i = 0$ means tail. Now we are ready to use Hoeffding's inequality, which says for independent variable $X_1, \ldots, X_n$ and $X_i \in [a_i, b_i]$, the sum of these variables S has the following inequality

$$\Pr(S - \mathbb{E}(S) \geq t) \leq \exp\left\{ -\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2} \right\}$$

Here in our case, $a_i = 0$ and $b_i = 1$ for any i in $\{1, \ldots, 2n\}$. Also we know $\mathbb{E}(X) = 2n \cdot \frac{1}{2} = n$. If we let $t = \frac{c}{2}\sqrt{n}$, we can write Hoeffding's inequality as

$$\Pr\{X_H - n \geq \frac{c}{2}\sqrt{n}\} \leq \exp\left\{ -\frac{2t^2}{2n} \right\} = \exp\left\{ -\frac{(c/2\sqrt{n})^2}{n} \right\} = \exp\left\{ -\frac{c^2}{4} \right\}$$

Comparing with original inequality, we just let $\varepsilon = -\frac{c^2}{4}$, and solve $c = -2\ln\varepsilon$. hence we proved for any $\epsilon > 0$ there exist such a constant c that satisfy the original inequality.

So c is a function of ε, which means a smaller ε requires a larger constant c. That also means, the probability of big difference between H and T is small.

8 Question 8

References

[1] Cristina Bazgan, Miklos Santha, and Zsolt Tuza. Efficient approximation algorithms for the subset-sums equality problem. In Kim Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 387–396. Springer Berlin / Heidelberg, 1998. URL `http://dx.doi.org/10.1007/BFb0055069`. 10.1007/BFb0055069.