15-451/651 Algorithms, Spring 2019
Recitation #9 Worksheet

Approximation Algorithms: First-Fit

You are packing up n items into boxes, and want to use as few boxes as possible. Each box can fit a total of 10 pounds of stuff, and the weight of the i^{th} item is $w_i \leq 10$. Your algorithm is this: initially, all the boxes are lined up, empty. You pick the next unpacked item (say item i), and put it in the *first* box that can hold the item (i.e., whose current weight is at most $10 - w_i$).

1. Argue that if OPT is the optimal number of boxes into which you can pack all the items, then your algorithm uses at most $2 \cdot OPT + 1$ bins.

   Solution: First, $OPT \geq \frac{1}{10} \sum_i w_i$, just by the weight restriction.

   Next, there cannot be two non-empty bins ending ≤ 5 pounds in our solution. Indeed, the items in the higher numbered bin could have been put into the lower numbered bins instead (they fit, because $5 + 5 \leq 10$). So at most one of our bins has ≤ 5, and the others are all more than 5. The latter number is $\leq 2OPT$ by the calculation above, and the former adds a 1.

2. Suppose you change your algorithm to the following greedier procedure (called *next-fit*): if the next unpacked item does not fit into the current box, then close the current box (never to look at it again), and open a new box. Show the same $2 \cdot OPT + 1$ guarantee for this algorithm.

   Solution: Almost the same analysis, use that any two *consecutive* boxes have total weight at least 10.

Linear Programs and Approximation Algorithms

In lecture you saw how to obtain a 2-approximation for vertex cover by (a) writing the problem as an integer linear program, (b) "relaxing" the integer constraints (of the form $x_v \in \{0, 1\}$) to fractional constraints (of the form $0 \leq x_v \leq 1$), and then (c) "rounding" the fractional solution to get an integer solution. Here is an example with Set Cover.

You want to do n tasks. You must hire some subset of m specialists $S_1, S_2, \ldots S_m$ to do all of them. Each specialist is capable of doing some subset of the tasks $[n]$. For each task, there is least one and at most k of these specialists capable of doing it. (Once you hire a specialist she can do all the tasks she is capable of doing.) You want to choose the smallest number of specialists to finish all tasks.

1. Write the problem as a integer linear program, with variable $x_i \in \{0, 1\}$ for each specialist S_i.

Solution:

$$\begin{aligned} ILP^* &= \min \textstyle\sum_{i=1}^m x_i & \\ \textstyle\sum_{i:S_i \text{ can do task } t} x_i &\geq 1 & \forall t \in [n] \\ x_i &\in \{0,1\}. & \end{aligned}$$

2. Replace the integer constraints with $0 \leq x_i \leq 1$. How does the optimal LP value relate to the original ILP value?

   Solution:

$$LP^* = \min \textstyle\sum_{i=1}^m x_i \tag{1}$$

$$\textstyle\sum_{i:S_i \text{ can do task } t} x_i \geq 1 \qquad \forall t \in [n] \tag{2}$$

$$0 \leq x_i \leq 1. \tag{3}$$

   Since we have less restrictive constraints, $LP^* \leq ILP^*$.

3. Given a fractional solution x^* to the LP above, can you suggest how to get an integer solution of at most k times LP^*?

   Solution: Since there are at most k specialists capable of doing task t, at least one specialists has $x_i^* \geq 1/k$. So if you pick all specialists with $x_i^* \geq 1/k$, you have a feasible solution, every task will be done. And the number of specialists you pick is at most $k \cdot LP^* \leq k \cdot ILP^* = k \cdot OPT$.

Online Algorithms: Searching for Keys

You live at position 0 of the PA turnpike. You've dropped your keys somewhere (at an integer location) and need to find them. The keys are small, so you cannot see them from afar, you will only see them when you are at the same location as them. If the keys are at location $X \in \mathbf{R}$, then the optimal solution is just to go to that location, and travel $|X|$. If you knew the sign of X, you could do that. But you don't.

What should you do to minimize the distance traveled? Your algorithm should have constant competitive ratio, ideally 9.

Solution: Do "doubling search". Go to positions $+1$, -2, $+4 = 2^2$, $-8 = -2^3$, $+16 = 2^4$, etc, until you find the keys. Suppose X lies in $(2^{2i}, 2^{2i+2}]$. Then the total distance you travel is

$$2(1 + 2 + 4 + \ldots + 2^{2i} + 2^{2i+1}) + X = 2 \cdot (2^{2i+2} - 1) + X \leq 9X.$$

For negative X, the calculation is similar. Indeed, if $X \in [-2^{2i+1}, -2^{2i-1})$. Then the total distance you travel is

$$2(1 + 2 + 4 + \ldots + 2^{2i-1} + 2^{2i}) + |X| = 2 \cdot (2^{2i+1} - 1) + |X| \leq 9|X|.$$

Online Algorithms: Examples for Non-MTF Algorithms

Give request sequences for the List Update problem where the following algorithms have competitive ratio $\Omega(n)$.

1. **Do Nothing:** Don't reorder the list.
2. **Single Exchange:** After accessing x, if x is not at the front of the list, swap it with its neighbor toward the front.
3. **Frequency Count:** Maintain a frequency of access for each item. Keep the list ordered by non-increasing frequency from front to back.

Solution: Say the starting list is $1, 2, \ldots, n$. Then

1. $n, n, \ldots$. If the number of requests T is very large, opt moved n to the front and $n + T$ for T requests, whereas we pay nT. So the competitive ratio is $\frac{nT}{n+T}$, which is $\Omega(n)$ when $T \geq n$.
2. $n, n-1, n, n-1, \ldots$. OPT will move both to the front, and pay $O(n)$ for this, and $O(1)$ per request after that. We will pay $\Omega(n)$ each time.
3. $1^T 2^T 3^T \ldots n^T$. The list does not change, so we will pay $T + 2T + \ldots + nT = \Theta(n^2 T)$. The optimal solution is to move each element to the front the first time we access it, so we pay $\Theta(n^2 + nT)$ in total. Hence the competitive ratio is $\Omega(n)$ when $T \geq n$.