Homework 11

Aleck Zhao

December 7, 2017

1 Bin Packing (50 points)

Suppose that we are given a set of n objects, where the size s_i of the ith object satisfies $0 < s_i < 1$. We wish to pack all the objects into the minimum number of unit-size bins. Each bin can hold any subset of the items whose total size does not exceed 1. In other words, we are trying to find a function $f : [n] \to [k]$ so that $\sum_{i:f(i)=j} s_i \leq 1$ for all $j \in [k]$, and our goal is to minimize k. Let $S = \sum_{i=1}^{n} s_i$.

(a) Prove that the optimal number of bins required is at least $\lceil S \rceil$.

Proof. If the optimal number of bins was less than $\lceil S \rceil$, then the total amount of space in the bins would be $\lceil S \rceil - 1$. If S is an integer, then this is $S - 1 < S$, which would not be enough space to hold every object. If S is not an integer, then this is $\lfloor S \rfloor < S$, and would not be enough space. Contradiction, so the optimal number of bins is at least $\lceil S \rceil$. □

The *first-fit algorithm* considers each object in turn (from 1 to n) and places it in the first bin that can accommodate it. If there is no such bin, then we create a new bin for it and make it the last bin. Note that this defines an ordering over bins based on when we created them, so "first" and "last" make sense.

(b) Prove that the first-fit algorithm leaves at most one bin at most half full. In other words, all bins but one are more than half full.

Proof. Suppose at termination there are at least 2 bins that are at most half full. Suppose two of them are B_1 and B_2, with B_1 created before B_2. Let x_i be the first object that was added to B_2 and let s_i be its size. Then $s_i \leq 0.5$ since the total size of B_2 was at most 0.5 at termination. Since the total size of B_1 at termination was at most 0.5, that means the size of B_1 at that iteration was at most 0.5, so x_i would have fit in B_1. Since B_1 was created before B_2, the algorithm would have placed x_i in B_1 instead of B_2. Contradiction, so at termination there is at most 1 bin that is at most half full. □

(c) Prove that the first-fit algorithm is a 2-approximation to bin packing.

Proof. If the algorithm produces m bins at termination, then by part (b), at least $n - 1$ are more than half full. Thus, the total size of all the objects is at least

$$\sum_{i=1}^{n} s_i = S > \frac{1}{2}(n-1) \implies 2S > n-1 \implies 2S \geq n \implies S \geq \frac{n}{2}$$

Since the optimal number of bins is at least S, we have

$$\frac{ALG}{OPT} \leq \frac{n}{n/2} = 2$$

so this is a 2-approximation, as desired. □

2 Pirate Treasure (50 points)

You are currently standing on an east-west road next to a sign. This sign tells you that somewhere along the road there is a pirate treasure that is sitting in plain sight and yet which is currently unclaimed. Unfortunately, the sign does not say whether the treasure is to the east of the sign or to the west, and also does not tell you the distance from the treasure. Your goal is to find the treasure while minimizing the amount that you travel. Let x denote the distance from the sign to the treasure (which you do not know). So if you walk in the correct direction and never turn around, your cost will be x (since you will find the treasure after walking distance x). However, if you walk in the *incorrect* direction and never turn around, then your cost will be infinite (since you will keep walking forever). You may assume that x is an integer. T

(a) Design a deterministic $O(1)$-competitive algorithm for this problem. That is, design an algorithm which finds the treasure and involves walking a total distance of at most $O(1) \cdot x = O(x)$. Prove the competitive ratio of your algorithm.

Solution. Consider the algorithm that begins by walking 1 unit east. If the treasure is not found, turn around back to the center, and walk 2 units west. If the treasure is not found, turn around back to the center, and walk 4 units east, and continue this process. Thus, at iteration $2i$, you move as far east as 2^{2i} units, and at iteration $2i+1$, you move as far west as 2^{2i+1} for $i \geq 0$.

If $2^{2i} < x \leq 2^{2i+2}$ east for some i, then you will proceed by the algorithm until you walk out to 2^{2i+1} west, then walk out to x east. The total distance traveled is

$$2(1 + 2 + \cdots + 2^{2i+1}) + x = 2(2^{2i+2} - 1) = 2^{2i+3} - 2 + x < 8x - 2 + x = O(x)$$

If $2^{2i-1} < x \leq 2^{2i+1}$ west for some i, then you will proceed by the algorithm until you walk out to 2^{2i} east, then walk out to x west. The total distance traveled is

$$2(1 + 2 + \cdots + 2^{2i}) + x = 2(2^{2i+1} - 1) + x = 2^{2i+2} - 2 + X < 8x - 2 + x = O(x)$$

Thus, the total distance traveled is $O(x)$, so the algorithm is $O(1)$-competitive. □

Now suppose that instead of the sign being located somewhere along an east-west road, it is located at an intersection where you have m directions to choose from (so m was 2 in the previous part). The sign tells you that the treasure is somewhere along one of the roads, but doesn't tell you which road or the distance to the treasure.

(b) Design a deterministic $O(m)$-competitive algorithm for this problem (and prove that it is $O(m)$-competitive).

Solution. Let the directions be numbered 0, 1, ..., $m-1$. Then consider the algorithm that examines the directions in order from 0 to $m-1$ then wrapping around to 0 and continuing. Then at iteration i, we are examining along direction $i \pmod m$ for a distance of $\left(1 + \frac{1}{m-1}\right)^i$, similar to part (a).

The worst case is when the treasure is just out of reach of some iteration k, so when $x = \left(1 + \frac{1}{m-1}\right)^k + \varepsilon$. Then it won't be found in that iteration, and will take until iteration $k+m$ to find again. The total amount of searching will be

$$\sum_{i=0}^{k+m} \left(1 + \frac{1}{m-1}\right)^i + \varepsilon = \frac{\left(1 + \frac{1}{m-1}\right)^{k+m+1} - 1}{\left(1 + \frac{1}{m-1}\right) - 1} + \varepsilon = m\left(1 + \frac{1}{m-1}\right)^{k+m} - m + 1 + \varepsilon$$

The optimal algorithm will search along the correct direction and take $x = \left(1 + \frac{1}{m-1}\right)^k + \varepsilon$, so the competitive ratio is

$$\frac{ALG}{OPT} = \frac{m\left(1 + \frac{1}{m-1}\right)^{k+m} - m + 1 + \varepsilon}{\left(1 + \frac{1}{m-1}\right)^k + \varepsilon} = O\left(m\left(1 + \frac{1}{m-1}\right)^m\right)$$

Since $\left(1 + \frac{1}{m-1}\right)^{m-1} \approx e$, it follows that the competitive ratio is $O(m)$. □

(c) Prove that for any deterministic algorithm, there is an input (location of the treasure) so that the competitive ratio of the algorithm is $\Omega(m)$.

Proof. Suppose the algorithm examines directions sequentially $d_0, d_1, \cdots$ with $d_i \in \{0, 1, \cdots, m-1\}$ and for distances $x_0, x_1, \cdots$. If some direction is never examined in the algorithm, placing the input along that direction leads to a competitive ratio of ∞.

Otherwise, every direction is examined at least once in the algorithm. WLOG 0 is the last examined direction at iteration m. Then consider

$$x = \min_{1 \le i \le m-1} \sum_{k:d_k=i}^{k<m} x_k$$

as the minimum of the total distance examined along each direction before direction 0 is examined. Then let the treasure be a distance x in direction 0. The total distance examined before direction 0 is examined is at least $(m-1)x$ and at least x more must be examined along direction 0 to find the treasure, so the total distance examined is at least mx. The optimal algorithm takes only x, so the competitive ratio is at least $\Omega(m)$. □