CS5234: Combinatorial and Graph Algorithms	Lecture 3

Tutorial: Week 3

Lecturer: Seth Gilbert *August 30, 2018*

1 Yao's Principle: Three Examples

Yao's Principle says that in order to show a worst-case running time for every randomized algorithm, it is sufficient to give an input distribution for which every deterministic algorithms performs badly. Or, more formally, let R be the class of randomized algorithms that solves a given problem and D be the class of deterministic algorithms that solve (the same) problem. Let X be the set of inputs (for both algorithms in R and X), and let γ be a specific distribution over the inputs in X. Then:

$$\forall A \in R \ : \ \max_{x \in X} \left(\mathrm{E}\left[cost(A, x)\right]\right) \geq \min_{B \in D} \left(\mathrm{E}\left[cost(B, x \text{ chosen from } \gamma)\right]\right) \ .$$

Here $cost(A, x)$ is the time that A takes when it is run on input x. The expectation on the left is over the random choices made by the algorithm $A \in R$, and the left-hand side is the worst-case expected cost for all randomized algorithms. The expectation on the right is over the choice of x from the distribution γ, and the right-hand side is the minimum expected cost of any deterministic algorithm when the input is chosen according to the distribution.

1.1 Example 1. Sorting

Show that comparison-based sorting requires expected $\Omega(n \log n)$ comparisons using Yao's Principle.

For the purpose of analyzing sorting, we have to give an input distribution such that every deterministic algorithm has expected number of comparisons $\Omega(n \log n)$.

We choose the distribution γ that selects each permutation of the integers from $\{1, \ldots, n\}$ uniformly with probability $1/n!$. We need to argue that every deterministic algorithm requires expected $\Omega(n \log n)$ comparisons.

Fix a deterministic sorting algorithm $B \in D$. Recall that each deterministic sorting algorithm can be represented as binary decision tree. Each leaf in the decision tree represents a permutation of the input, i.e., each input permutation terminates at a different leaf. Overall, the decision tree for B has $n!$ leaves. Since our chosen input distribution selects a random permutation, the algorithm B will terminate at a randomly chosen leaf. Thus, the expected number of comparisons is equal to the depth of a randomly chosen leaf.

Recall that there are $n!$ leaves in total. Let us look at the depth of the $n!/2$ leaves with the lowest depth. Since a binary tree with x leaves must have height at least $\log(x)$, we conclude that the $n!/2$ leaves with the lowest depth must be part of a subtree of depth at least:

$$\begin{aligned} \log(n!/2) &= \log(n!) - 1 \\ &\geq \ln(n!) - 1 \\ &\geq n\ln(n)/2 - 1 \\ &\geq n\ln(n)/4 \end{aligned}$$

This follows from Sterling's approximation which states that $\ln(n!) = n\ln(n) - n + O(\ln(n))$, as long as $n > 8$.

Since we know that the lowest depth $n!/2$ leaves have depth at least $n\ln(n)/4$, this means we conclude that there are at least $n!/2$ leaves of depth $> n\ln(n)/4$. Therefore, if we choose a leaf at random, the expected depth will be at least $[(n\ln(n)/4)(n!/2) + (0)(n!/2)]/n! = n\ln(n)/8$. (This is obviously an underestimate as it assumes that the smallest depth $n!/2$ leaves have depth 0.)

We conclude that the expected number of comparisons for algorithm B is $\Omega(n \log n)$. By applying Yao's Principle, we conclude that every randomized comparison-based sorting algorithm takes time $\Omega(n \log n)$.

1.2 Example 2. Property Testing

Given a binary array $A[1, n]$ where $A[i] \in \{0, 1\}$, show that it requires time at least $\Omega(1/\epsilon)$ to decide whether array A is all-zero or ϵ-far from all zero with probability at least $2/3$.

First, we specify an input distribution. Assume (for simplicity) that n is a multiple of $1/\epsilon$. We divide the array into $1/\epsilon$ chunks of size ϵn. We choose one of these chunks uniformly at random and set every array position in the chunk to 1; we set all the remaining array positions in the array (in all the other chunks) to zero. Notice that this array is ϵ-far from all-zero as there are ϵn array slots set to one. Now, with probability $1/2$ we choose the array just constructed, and with probability $1/2$, we choose the all-zero array. This construction defines a distribution over inputs to the algorithm.

We need to show that any deterministic algorithm B that access the array $1/(3\epsilon)$ times will fail to correctly classify this input with probability at least $2/3$.

Fix some deterministic algorithm B. Notice that B accesses only one-third of the chunks. There are two cases. First, if B only accesses slots containing zero, it may output *far from all-zero*. In this case, algorithm B is wrong with probability $1/2$, i.e., all the times that out input distribution selects the all zero array.

Alternatively, if B only accesses slots containing zero, it may output *all zero*. In this case, with probability $1/2$ our input distribution selects an array that is far from all-zero. Since B only accesses one-third of the chunks, it only sees a one with probability at most $1/3$. That is, with probability $\geq 2/3$, algorithm B sees only zeros. Thus with probability $(1/2)(2/3) = 1/3$, algorithm B outputs *all zero*.

Thus, in either case, algorithm B fails with probability at least $1/3$, as required. By Yao's Principle, this implies that every randomized algorithm requires at least $1/(3\epsilon)$ array accesses to different the all-zero array from an array ϵ-far from all-zero.

Notice that here we used a slight variant of Yao's Principle, in that we did not analyze the expected running time. Instead, we used the version that was presented in Problem Set 2, i.e.:

Theorem 1 (Yao's Principle) *Assume the following:*

> *There exists a distribution D of the inputs such that: for every deterministic algorithm A of query complexity q, $Pr[A(x)$ is wrong$] > 1/3$.*

Then we can conclude:

> *For any randomized algorithm A of query complexity q there exists an input x such that: $Pr[A(x)$ is wrong$] > 1/3$.*

1.3 Example 3. Approximate Minimum Spanning Tree

Show that every randomized algorithm that finds a sufficiently good additive approximation to the MST weight with probability at least $2/3$ requires at least $\Omega(W)$ time.

See the solutions to Problem Set 2.