Lecture Notes: The Core of Backward Analysis

Yufei Tao
Department of Computer Science and Engineering
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

March 12, 2024

Backward analysis is a technique often deployed to bound the expected running time of randomized algorithms. In this lecture, we will discuss its core ideas through a contrived example. The next few lectures will apply the technique to deal with non-trivial computational geometry problems.

Preliminaries. We will start by reviewing a rudimentary fact about probabilities. Let A be an arbitrary event, and let $B_1, B_2, ..., B_k$ (for any integer $k \geq 2$) be mutually disjoint events that form a partition of the probability space. It holds that

$$\mathbf{Pr}[A] \;=\; \sum_{i=1}^{k} \mathbf{Pr}[A \mid B_i] \cdot \mathbf{Pr}[B_i]. \tag{1}$$

For example, consider rolling a dice (with six facets) 10 times, each time getting a number from 1 to 6. Let A be the event "the 10 numbers obtained add up to at least 30". Set $k = 11$ and for each $i \in [0, 10]$, define B_i be the event "among the 10 numbers obtained, i of them are 1". The events $B_0, B_1, ..., B_{10}$ are mutually disjoint and cover the whole probability space because exactly one of those events must occur in any case. From (1), we have $\mathbf{Pr}[A] = \sum_{i=0}^{10} \mathbf{Pr}[A \mid B_i] \cdot \mathbf{Pr}[B_i]$.

A "Silly" Algorithm. Suppose that, given a set S of $n \geq 3$ integers, our goal is to compute its *minimum bounding interval* (MBI) $[x, y]$, namely, x (resp., y) is the smallest (resp., largest) integer of S. This can be trivially done in $O(n)$ time, but our purpose is to illustrate backward analysis through the following algorithm.

algorithm silly-MBI

1. randomly permute S and store the result in array A
   /* this can be done in $O(n)$ time; see the appendix */
2. $x \leftarrow \min\{A[1], A[2]\}$ and $y \leftarrow \max\{A[1], A[2]\}$
3. **for** $i = 3$ **to** n **do**
4.   **if** $A[i] \in [x, y]$ **then continue**
       else
5.   set $x \leftarrow \min\{A[1], A[2], ..., A[i]\}$ by scanning $A[1], A[2], ..., A[i]$ again
6.   set $y \leftarrow \max\{A[1], A[2], ..., A[i]\}$ by scanning $A[1], A[2], ..., A[i]$ again
           /* note: Lines 5-6 cost $O(i)$ time */

What a silly algorithm! Clearly, it requires $O(n^2)$ time in the worst case. At Line 5, any smart student will choose to update x as $\min\{x, A[i]\}$, which takes only $O(1)$ time, as opposed to $O(i)$; a similar statement can be said about Line 6. These simple changes will allow you to reduce the time complexity from $O(n^2)$ to $O(n)$!

Perhaps you would be surprised to learn that the silly-MBI algorithm is not too bad after all: its *expected* running time is $O(n)$! Next, we will use backward analysis to prove it.

Analysis. We will focus on Lines 3-6, which perform an iteration for each $i \in [3, n]$. The iteration for i takes

- $O(1)$ time if $A[i] \in [x, y]$ — in this case, we call this a *lucky* iteration;
- or $O(i)$ time otherwise — in this case, we call this an *unlucky* iteration.

Let us introduce a random variable:

$$X_i = \begin{cases} 1 & \text{if the iteration of } i \text{ is unlucky} \\ 0 & \text{if the iteration of } i \text{ is lucky} \end{cases}$$

Thus, the iteration of i has an expected running time of $O(1) + \mathbf{Pr}[X_i = 1] \cdot O(i)$. Hence, we can bound the overall expected cost of Lines 3-6 as

$$\sum_{i=3}^{n} O(1) + \mathbf{Pr}[X_i = 1] \cdot O(i). \tag{2}$$

We will prove:

Lemma 1. $\mathbf{Pr}[X_i = 1] = 2/i$.

Given the above lemma, the expected cost of silly-MBI in (2) evaluates to $O(n)$, as claimed. Before proving the lemma in general, let us discuss the special case of $i = n$: how to analyze $\mathbf{Pr}[X_n = 1]$? For this purpose, observe that the iteration of $i = n$ is unlucky if and only if $A[n]$ is either the smallest or largest element of S. Due to random permutation (performed at Line 1 of silly-MBI), every element of S has the same chance to sit at $A[n]$. It thus follows that $\mathbf{Pr}[X_n = 1] = 2/n$.

We are now ready to prove Lemma 1 for any $i \in [3, n-1]$. Define a random variable

$$\Pi = \text{the sequence } A[i+1], A[i+2], ..., A[n]$$

Because the permutation is random, Π can take $n!/i!$ possible "values", each being a different way to permute $n-i$ elements of S; furthermore, each of those "values" occurs with the same probability. To put this formally, let π be any possible permutation of $n-i$ elements of S, it holds that

$$\mathbf{Pr}[\Pi = \pi] = i!/n!.$$

We now apply (1) to derive

$$\begin{aligned} \mathbf{Pr}[X_i = 1] &= \sum_{\text{all } \pi} \mathbf{Pr}[X_i = 1 \mid \Pi = \pi] \cdot \mathbf{Pr}[\Pi = \pi] \\ &= \frac{i!}{n!} \sum_{\text{all } \pi} \mathbf{Pr}[X_i = 1 \mid \Pi = \pi] \qquad (3) \end{aligned}$$

We will argue shortly that $\mathbf{Pr}[X_i = 1 \mid \Pi = \pi]$ is always $2/i$, with which we obtain

$$(3) = \frac{i!}{n!} \sum_{\text{all } \pi} \frac{2}{i} = 2/i$$

where the last step used the fact that there are $n!/i!$ different π.

All that remains is to prove $\mathbf{Pr}[X_i = 1 \mid \Pi = \pi] = 2/i$. This, in fact, is no more complicated than proving $\mathbf{Pr}[X_n = 1] = 2/i$. Denote by $S \setminus \pi$ the set of elements in S that are outside π, namely, $S \setminus \pi = \{e \in S \mid e \notin \pi\}$. Under the event $\Pi = \pi$, the iteration of i is unlucky if and only if $A[i]$ is the smallest or largest element of $S \setminus \pi$. It thus follows that $\mathbf{Pr}[X_i = 1 \mid \Pi = \pi] = 2/i$.

The approach we used to prove $\mathbf{Pr}[X_i = 1]$ is what is known as *backward analysis.*

Random Permutation. Let A be an array of n elements. The following algorithm computes a random permutation of A:

algorithm permute

1. **for** $i = 2$ **to** n **do**
2. $\quad x \leftarrow$ a random number in $[1, i]$
3. $\quad$ swap $A[i]$ and $A[x]$
   $\quad$ /* note: the swap has no effect if $i = x$ */

The algorithm finishes in $O(n)$ time.