Chapter 33

Introduction

This chapter introduces randomized algorithms and presents an overview of the techniques used for analyzing them.

1 Randomized Algorithms

Definition 33.1 (Randomized Algorithm)**.** We say that an algorithm is ***randomized*** if it makes random choices. Algorithms typically make their random choices by consulting a ***source of randomness*** such as a (pseudo-)random number generator.

Example 33.1. A classic randomized algorithm is the quick-sort algorithm, which selects a random element, called the pivot, and partitions the input into two by comparing each element to the pivot. To select a randomly chosen pivot from n elements, the algorithm needs $\lg n$ bits of random information, usually drawn from a pseudo-random number generator.

Definition 33.2 (Las Vegas and Monte Carlo Algorithms)**.** There are two distinct uses of randomization in algorithms.

The first method, which is more common, is to use randomization to weaken the cost guarantees, such as the work and span, of the algorithm. That is, randomization is used to organize the computation in such a way that the impact is on the cost but not on the correctness. Such algorithms are called ***Las Vegas algorithms***.

Another approach is to use randomization to weaken the correctness guarantees of the computation: an execution of the algorithm might or might not return a correct answer. Such algorithms are called ***Monte Carlo algorithms***.

Note. In this book, we only use Las Vegas algorithms. Our algorithm thus always return the correct answer, but their costs (work and span) will depend on random choices.

Random Distance Run. Every year around the middle of April the Computer Science Department at Carnegie Mellon University holds an event called the "Random Distance

Run". It is a running event around the track, where the official die tosser rolls a die immediately before the race is started. The die indicates how many initial laps everyone has to run. When the first person is about to complete the laps, the die is rolled again to determine the additional laps to be run. Rumor has it that Carnegie Mellon scientists have successfully used their knowledge of probabilities to train for the race and to adjust their pace during the race (e.g., how fast to run at the start).

Thanks to Carnegie Mellon CSD PhD Tom Murphy for the design of the 2007 T-shirt.

1.1 Advantages of Randomization

Randomization is used quite frequently in algorithm design, because of its several advantages.

- **Simplicity:** randomization can simplify the design of algorithms, sometimes dramatically.
- **Efficiency:** randomization can improve efficiency, e.g., by facilitating "symmetry breaking" without relying on communication and coordination.

- **Robustness:** randomization can improve the robustness of an algorithm, e.g., by reducing certain biases.

Example 33.2 (Primality Testing)**.** A classic example where randomization simplifies algorithm design is primality testing. The problem of ***primality testing*** requires determining whether a given integer is prime. By applying the theories of primality developed by Russian mathematican Arthujov, Miller and Rabin developed a simple randomized algorithm for the problem that only requires polynomial work. For over 20 years it was not known whether the problem could be solved in polynomial work without randomization. Eventually a polynomial time algorithm was developed, but it is more complex and computationally more costly than the randomized version. Hence in practice everyone still uses the randomized version.

Definition 33.3 (Symmetry Breaking)**.** In algorithm design, the term ***symmetry breaking*** refers to an algorithm's ability to distinguish between choices that otherwise look equivalent. For example, parallel algorithms sometimes use symmetry breaking to select a portion of a larger structure, such as a subset of the vertices of a graph, by making local decisions without necessarily knowing the whole of the structure. Randomization can be used to implement symmetry breaking.

Example 33.3. Suppose that we wish to select a subsequence of a sequence under the condition that no adjacent elements are selected. For example, given the sequence

$$\langle\, 0, 1, 2, 3, 4, 5, 6, 7 \,\rangle\,,$$

we could select

$$\langle\, 0, 2, 4, 6 \,\rangle\,,$$

or

$$\langle\, 1, 3, 6 \,\rangle\,,$$

but not

$$\langle\, 0, 1, 3, 6 \,\rangle\,.$$

An algorithm can make such a selection by "flipping a coin" for each element and selecting the element if it has flipped heads and its next (following) neighbor has flipped tails.

Exercise 33.1. Prove that the algorithm described above is correct.

Exercise 33.2. What is the expected length of the selected subsequence in terms of the length of the input sequence?

Algorithmic Bias. As algorithms take on more and more sophisticated decisions, the questions of bias naturally arise. Deterministic algorithm are particularly susceptible to creating bias when they make decisions based on partial or imperfect decisions and repeat that same decisions on many problem instances. Randomness can reduce such bias, e.g., by choosing one of the alternatives based on some probability distribution.

Example 33.4. Consider the algorithm operating a self driving vehicle. In certain circumstances, the algorithm might have to choose one of two actions none of which are desirable. If the algorithm makes its decisions deterministically, then it will repeat the same decision in identical situations, leading to a bias towards one of the bad choices. Such an algorithm could thus make biased decisions, which in total could lead to ethically unacceptable outcomes.

1.2 Disadvantages of Randomization

Complexity of Analysis. Even though randomization can simplify algorithms, it usually complicates their analysis. Because we only have to analyze an algorithm once, but use it many times, we consider this cost to be acceptable.

Uncertainty. Randomization can increase uncertainity. For example, a randomized algorithm could get unlucky in the random choices that it makes and take a long time to compute the answer. In some applications, such as real-time systems, this uncertainty may be unacceptable.

Example 33.5. The randomized quicksort algorithm can perform anywhere from $\Omega(n^2)$ to $\Theta(n \lg n)$ work and a run of quicksort that requires $\Omega(n^2)$ work could take a very long time to complete. Depending on the application, it can therefore be important to improve the algorithm to avoid this worst case.

2 Analysis of Randomized Algorithms

Definition 33.4 (Expected and High-Probability Bounds)**.** In analyzing costs for randomized algorithms there are two types of bounds that are useful: expected bounds, and high-probability bounds.

- ***Expected bounds*** inform us about the average cost across all random choices made by the algorithm.
- ***High-probability*** bounds inform us that it is very unlikely that the cost will be above some bound. For an algorithm, we say that some property is true with ***high probability*** if it is true with probability $p(n)$ such that

  $$\lim_{n \to \infty} (p(n)) = 1,$$

  where n is an algorithm specific parameter, which is usually the instance size.

As the terms suggest, expected bounds characterize average-case behavior whereas high-probability bounds characterize the common-case behavior of an algorithm.

Example 33.6. If an algorithm has $\Theta(n)$ expected work, it means that when averaged over all random choices it makes in all runs, the algorithm performs $\Theta(n)$ work. Because expected bounds are averaged over all random choices in all possible runs, there can be runs that require more or less work. For example once in every $1/n$ tries the algorithm might require $\Theta(n^2)$ work, and (or) once in every $\sqrt{n}$ tries the algorithm might require $\Theta(n^{3/2})$ work.

Example 33.7. As an example of a high-probability bound, suppose that we have n experiments where the probability that work exceeds $O(n \lg n)$ is $1/n^k$. We can use the union bound to prove that the total probability that the work exceeds $O(n \lg n)$ is at most $n \cdot 1/n^k = 1/n^{k-1}$. This means that the work is $O(n \lg n)$ with probability at least $1 - 1/n^{k-1}$. If $k > 2$, then we have a high probability bound of $O(n \lg n)$ work.

Remark. In computer science, the function $p(n)$ in the definition of high probability is usually of the form $1 - \frac{1}{n^k}$ where n is the instance size or a similar measure and k is some constant such that $k \geq 1$.

Analyzing Expected Work. Expected bounds are quite convenient when analyzing work (or running time in traditional sequential algorithms). This is because the linearity of expectations allows adding expectations across the components of an algorithm to get the overall expected work. For example, if the algorithm performs n tasks each of which take on average 2 units of work, then the total work on average across all tasks will be $n \times 2 = 2n$ units.

Analyzing Expected Span. When analyzing span, expectations are less helpful, because bounding span requires taking the maximum of random variables, rather than their sum. And the expectation of the maximum of two randow variables is not equal to the maximum of expectations of the random variables. To bound the span, we will usually need stronger guarantees than expectation, typically in the form of high probability bounds. High-probability bounds allow us to bound the expectation of the maximum of a number of random variables by showing that it is highly unlikely for any one of them to be large.

Exercise 33.3. Consider a game in which we draw some number of tasks at random such that a task has length n with probability $1/n$ and has length 1 otherwise. The expected length of a task is therefore bounded by 2. Imagine now drawing n tasks and waiting for all them to complete, assuming that each task can proceed in parallel independently of other tasks. Prove that the expected completion time is not constant.

Note. The exercise corresponds closely to computing the span of a computation, because the time waited depends on the length of the longest task.

Exercise 33.4. Repeat the same exercise with slightly different probabilities: a randomly chosen task has length n with probability $1/n^3$ and 1 otherwise. Prove now that the expected completion time is bounded by a constant.