**CMPUT 675: Approximation Algorithms** **Fall 2011**

# Lecture 1 (Sep 8, 2005 ): A sample scribe

*Lecturer: Mohammad R. Salavatipour* *Scribe: Mohammad R. Salavatipour*

## 1.1 Introduction

What is a randomized algorithm? Put it simply, any algorithm that makes random choices during its execution. We assume these algorithm have access to a source of fair random bits. Why do we study and use randomized algorithms? There are several factors, some of the more important ones are:

**Speed:** Randomized algorithms are usually fast, sometimes much faster than any deterministic algorithm for the same problem. Even if the asymptotic running time is the same as the deterministic one it can be faster in practice. For example QuickSort, though has worse-case running time worse than that of MergeSort or Heapsort, it is the fastest algorithm in practice. The disadvantage of some randomized algorithms (like QuickSort) is that sometimes the running time is more than usual. Also, some randomized algorithms produce a solution that is correct with high (but smaller than 1) probability, so there is a chance that the solution is not correct.

**Simplicity:** Even if the known randomized algorithm is not faster than the deterministic one, it is often simpler.

**Only thing we can do:** In some situations, randomized algorithms are the only algorithms we know. For example, in some protocols or online algorithms (in the presence of an adversary) randomness helps to defeat the adversary, whereas any deterministic algorithm can be defeated by an adversary. Or, to check if a multivariable polynomial is identical to zero we only know randomized algorithms. Note that the description of the polynomial might be implicit, e.g. the determinant of a matrix whose entries contain different variables.

Randomized algorithms should not be confused with probabilistic analysis of algorithms and average case analysis: in the average case analysis we assume some probability distribution (e.g. uniform) on input and then analyze the running time based on the assumption that the input is drawn from that probability distribution (e.g. average case analysis of QuickSort). However, when we analyze randomized algorithms we do not assume a particular (random) distribution on input. The analysis must hold for all inputs.

## 1.2 An Example: Comparing strings with communication costs

Suppose that Alice and Bob each have an $n$-bit binary string, call them $a$ (for Alice) and $b$ (for Bob). They want to find out if these two strings are equal. There is a communication link between them over which they can send bits. It is enough if one of them figures out whether $a = b$ or not. However, there is a cost associated with each bit transmission. Clearly, any deterministic algorithm needs to use $n$ bits of communication (Why?). Here we give a randomized algorithms which uses only $O(\log n)$ bits and has a very high success probability. The idea behind the solution is general: instead of comparing two objects from a large universe, map them into two objects from a smaller universe and then compare those two. These two smaller objects behave as the fingerprint of the two larger ones.

We will fix the number $t$ later. Alice picks a prime number $p \leq t$ uniformly at random and then sends both $p$ and $a \bmod p$ to Bob. Bob computes $b \bmod p$ and if it is equal to $a \bmod p$ then says they are equal. Otherwise he concludes that they are not equal.

Clearly if $a = b$ then this protocol gives the correct answer (since $a \bmod p = b \bmod p$). However, if $a \neq b$ there is a chance that this algorithm give a wrong answer. We upper bound this probability (and therefore find an upper bound on the error probability of the algorithm).

**Lemma 1** *With $t = c \cdot n \ln n$ for some large constant $c$:* $\Pr[a \bmod p = b \bmod p] \in O(\frac{1}{c})$.

**Proof.** Since $|a - b| \leq 2^n$ and since every prime is at least 2, there are at most $n$ primes that divide $|a - b|$. This is an upper bound on the number of "dangerous" primes, i.e. those that if we select the algorithm will fail. The Prime Number Theorem says that the number of primes up to $x$ is about $\frac{x}{\ln x}$. Therefore, there are $\frac{t}{\log t}$ primes that we can choose, out of which at most $n$ are dangerous. Thus:

$$\begin{aligned}
\Pr[failure] = \Pr[a \bmod p = b \bmod p] &= \Pr[p \text{ divides } |a-b|] \\
&\leq \frac{n}{\frac{t}{\ln t}} \\
&= \frac{n[\ln n + \ln\ln n + \ln c]}{c \cdot n \ln n} \in O(\frac{1}{c})
\end{aligned}$$

■

Therefore, the algorithm succeeds with probability at least $1 - \epsilon$ where $\epsilon$ is made arbitrary close to 0 by making $c$ large enough.

## 1.3 Some formulas and Algorithms

Here is a linear program:

$$\begin{array}{llrcll}
\textbf{LP-SLST} & \min & \sum_e c(e) \cdot x_e & & & \\
& \text{s.t.} & \sum_{p \in P_t | e \in p} f(p) & \leq & x_e & \forall e \in E, \quad t \in T \quad (1) \\
& & \sum_{p \in \mathcal{P}_t} f(p) & \geq & 1 & t \in T \quad (2) \\
& & x_e, f(p) & \geq & 0 & \forall e \in E, \quad p \in \cup_t \mathcal{P}_t \quad (3)
\end{array}$$

### 1.3.1 Set Cover: Greedy and randomized rounding

Here is a greedy algorithm for set cover:

Now we use randomized rounding algorithm to solve Set Cover. First consider the following simple algorithm:

1. Compute an optimal solution $x^*$ to the LP relaxation. Let $z^*$ be the value of this optimal solution.
2. For each set $S_i$, pick $S_i$ with probability $x_i^*$; in other words let $\hat{x}_i = 1$ be 1 with probability $x_i^*$ and zero with probability $1 - x_i^*$. Therefore, $\hat{x}_i$ is the rounded value of $x_i^*$.
3. Return the collection $C = \{S_j | \hat{x}_j = 1\}$

**SetCover Greedy Algorithm**

**Input:** Uinversal set $U = \{e_1, e_2, \ldots, e_n\}$, subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, with each $S_i \subseteq U$ having cost $C_i$.
**Output:** A minimum cost subset $S' \subseteq \mathcal{S}$ such that each $e_i \in U$ is in some $S_j \in S'$.
1. $C \leftarrow \emptyset$
2. $S' \leftarrow \emptyset$
3. **while** $C \neq U$ **do**
4. select $S_i \in S$ with minimum cost effectiveness $\alpha = \frac{c(S_i)}{|S_i - C|}$ with respect to $C$
5. for each $e \in S_i$, define $price(e)$ as $\alpha$
6. $S' \leftarrow S' \cup \{S_i\}$
7. $C \leftarrow C \cup S_i$
8. **return** $S'$

Figure 1.1: Algorithm SetCover

The expected value of the cost of the returned collection of selected sets is computed as follows:

$$\begin{aligned} E\left[cost(C)\right] &= \sum_{i=1}^{m} \Pr\left[S_i \text{ being selected}\right] \cdot c(S_i) \\ &= \sum_{i=1}^{m} x_i^* \cdot c(S_i) \\ &= z^* \end{aligned}$$

So the expected cost of the solution is the same as the optimum. However, this algorithm randomly picks sets of $S_j$ with a probability that is given by solving the problem using LP, so the solution might not be feasible. To address this, it's enough to repeat the Step 2, $4 \ln n$ times, and return the union of collection of all sets selected in all these repetitions. That is, if $C_i$ is the collection of sets that are selected at $i^{th}$ iteration of Step 2, then in Step 3, we would $C = \cup_{i=1}^{4 \ln n} c_i$. We show that this new algorithm has a good chance of returning a feasible solution with approximation ratio $O(\log n)$.

Consider an arbitrary element $e_i$ and WLOG assume that it belongs to sets $S_1, \ldots, S_k$. Since we start from a feasible solution, we must have $x_1^* + x_2^* + \ldots + x_k^* \geq 1$; in the worst case we should have $x_1^* + x_2^* + \ldots + x_k^* = 1$. What is the probability that none of the corresponding $k$ sets be selected given the probabilities of $x_i^*$'s? The following lemma (whose proof is starightforward) suggests that this probability is the highest when all the $x_i^*$'s are equal.

**Lemma 2** *Given $x_1 + \ldots + x_k = 1$, if we select each $S_i$ with probability $x_i^*$, the probability that no $S_i$ will be selected is highest (worst case) when all $x_i^* = \frac{1}{k}$.*

Using Lemma 2, we would like to compute the probability that randomized rounding would not be able to find a feasible solution. We take the worst case conditions into account in order to obtain an upper bound probability.

$$\Pr\left[e_i \notin c_j\right] \leq \left(1 - \frac{1}{k}\right)^k \leq e^{-1}$$

Therefore, for $4 \ln n$ iterations, we would have:

$$\Pr\left[e_i \in \cup_{j=1}^{4 \ln n} c_j\right] \leq \left(e^{-1}\right)^{4 \ln n} \leq \frac{1}{4n}$$

Therefore, using union bound and the fact that there are $n$ elements:

$$\Pr\left[\text{some } e_i \text{ is not covered}\right] \leq n \cdot \frac{1}{4n} \leq \frac{1}{4}.$$

Using the analysis done earlier, the expected cost of this solution is bounded by:

$$\mathrm{E}\left[cost(C)\right] \leq 4 \ln n \cdot z^*$$

Using Markov's inequality, $\Pr[X \geq t] \leq \frac{E[X]}{t}$:

$$\Pr\left[cost(C) > 4 \ln n \cdot z^*\right] \leq \frac{1}{4}$$

Therefore, with a probability greater than or equal to $\frac{1}{2}$, we have a feasible solution with cost less than or equal to $4 \ln n \cdot z^*$ which is in $O(\log n \cdot \mathrm{OPT})$, where OPT is the cost of the optimal solution. To get better probability it's enough to repeat the algorithm a constant number of times and get the best answer.