6.856 — Randomized Algorithms

David Karger

Handout #12, October 14, 2002 — Homework 5 Solutions

M. R. refers to this text:
Motwani, Rajeez, and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge: Cambridge University Press, 1995.

Problem 1 Randomized selection.

We will use the two-point sampling scheme described in class, which only requires $O(\log n)$ random bits, to choose the random elements needed for the selection algorithm. This sampling scheme assumed that the number of elements n was actually a prime. There are several way to circumvent this assumption for our problem. Perhaps the easiest is to increase n to a prime. To do this, identify a prime p, $n < p < 2n$ (this can done in linear time). Add $p - n$ new elements of value ∞ to the array, so that the k-th element of the original array is still the k-th element in the new array. Note that since we have at most doubled the number of elements, running in linear time on the new set implies we run in linear time on the original set.

So we can now assume without loss of generality that n is in fact a prime. We now apply the algorithm in the book. Since the book analysis only uses the Chebyshev bound, it works so long as the variables being analyzed are pairwise independent. Since the selections via two-point sampling are pairwise independent, we know that the variables X_i in the textbook analysis are pairwise independent. It follows that the algorithm failure probability is $O(n^{-1/4})$.

To decrease this failure probability, run that algorithm 4 times, using an independent $O(\log n)$-bit seed for the pairwise independent choices in each run. This still requires $O(\log n)$ random bits. This approach fails only if all 4 runs fail to find the k-th element, which happens with probability $O(1/n)$. Note this relies on the fact that in a single run of the original algorithm, we know whether or not we find the k-th element.

Problem 2 MR 5.12.

We observed in class that the size of the partition is equal to n plus the number of "split" events that occur when an edge crosses some autopartition line (cf. section 1.3 in the text). We build a permutation such that the algorithm that chooses split edges in this permutation order produces a tree of size $O(n \log n)$. We do so by walking down a decision tree where the i-th decision level corresponds to choosing which line appears at position i in the permutation. Suppose that we have already selected a prefix π_k of edges, say $\pi_k = e_1, \ldots, e_k$, to head the permutation and now want to select edge e_{k+1} from those remaining. Let S denote the number of split events that occur in our autopartition; we know

that when we select our permutation uniformly at random, $E[S] = O(n \log n)$. We also know by conditional expectations that

$$E[S \mid \pi_k] = \frac{1}{n-k} \sum_{e_{k+1}} E[S \mid \pi_k, e_{k+1}].$$

It follows that there always exists some e_{k+1} such that

$$E[S \mid \pi_k, e_{k+1}] \leq E[S \mid \pi_k].$$

If we always select such an extension, then after n steps we have a permutation π_n for which the value

$$E[S \mid \pi_n],$$

which is just the (deterministic) number of split events in π_n, is $O(n \log n)$.

In order to find the right permutation, we need only be able to compute $E[S \mid \pi_k]$ for any π_k. Let $v_1, \ldots, v_r = v$ be the set of edges that are on the line extending from u to v. If any of the v_i appear in the permutation before u, then they "protect" v from being split by u. Unfortunately, the converse is not true, since there might be edges not among the v_i whose *extension* protects v from u.

But assume that from now on we assume that extensions of edges do *not* protect other edges from being cut. This might make the BSP tree larger, because more splits occur. But the analysis given above and in class still applies, making the expected size $O(n \log n)$. So we can apply the method of conditional expectations to arrive at a BSP of size at most $O(n \log n)$.

If any of v_i appears in the permutation before u, then they "protect" v from being split by u. If none does, then v is split by u — we write $u \dashv v$. It follows that

$$E[S \mid \pi_k] = \sum_{u,v} \Pr[u \dashv v \mid \pi_k].$$

Therefore we only need to be able to compute

$$\Pr[u \dashv v \mid \pi_k]$$

in polynomial time. Now clearly,

$$\Pr[u \dashv v \mid \pi_k] = 0$$

if some v_i is in π_k and u is not, or if some v_i precedes u in π_k, while

$$\Pr[u \dashv v \mid \pi_k] = 1$$

if both u appears in π_k before all v_i, and finally

$$\Pr[u \dashv v \mid \pi_k] = 1/(r+1) \quad (*)$$

if none of u or $v_1, \ldots, v_r$ is in π_k (since the probability is $1/(r+1)$ that u will be picked before all the v_i).

Since we can compute these probabilities (it is trivial to find the intersection points of lines, and then sort the results), we can implement the conditional expectations algorithm.

Problem 3 MR 5.11.

(a) At a certain node of the conditional probability tree, consider a particular set for which k items remain to be placed. Depending on the items which have already been assigned values, there is a certain maximum number n_+ of items that can receive value $+1$ without breaking the balance limit of $O(\sqrt{n \log n})$ (by having too many positive items), and a certain minimum number n_- of items that can receive value $+1$ without breaking the limit (by having too few positive items). The probability that we get exactly j of our k values assigned $+1$ is just

$$\binom{k}{j} 2^{-k},$$

since each of the 2^k assignments is equally likely, and $\binom{k}{j}$ of them give k values $+1$. We simply need to sum this over $n_- \leq k \leq n_+$.

Thus, the probability that a particular row deviates from balance is a sum of binomial coefficients times a power of two, where the power of two is at most $n-i$ since $k \leq n-i$. But $\hat{P}(a)$ is just a sum of these sums.

(b) We work with binomial coefficients $\binom{k}{j}$ where $k \leq n$. So we start by pre-computing the $O(n^2)$ values $\binom{k}{j}$ in $O(n^2)$ time using Pascal's triangle (note that this uses only additions, no multiplications). Then computing $\hat{P}(a)$ is just some table lookups, shifts (for powers of two) and adds. In the unit-cost RAM, we are done. In the log-cost RAM, we must observe that the operands have size $O(\log \binom{n}{k}) = O(n)$ bits so that the individual addition operations take $O(n)$ time.

(c) Let $P_i(a)$ be the probability that row i deviates given a. We have $\hat{P}(a) = \sum P_i(a)$. Since P_i is a probability, we know that

$$P_i(a) = \frac{1}{2} P_i(b) + \frac{1}{2} P_i(c).$$

It follows that

$$\begin{aligned} \hat{P}(a) &= \sum P_i(a) \\ &= \sum \frac{1}{2} P_i(b) + \sum \frac{1}{2} P_i(c) \\ &= \frac{1}{2} \hat{P}(b) + \frac{1}{2} \hat{P}(c) \end{aligned}$$

The result follows.

(d) In unit cost RAM we use $O(n^2)$ time to pre-compute the binomial coefficients. Computing $P_i(a)$ then requires adding $O(n)$ binomial coefficients for work $O(n)$, which means $O(n^2)$ work suffices for $\hat{P}(a)$. We need to do this over $O(n)$ levels of tree for $O(n^3)$ total work. If we use a log-cost RAM, each addition takes $O(n)$ time (since all coefficients are $\leq 2^n$), so the total time rises to $O(n^4)$.

Problem 4 MR 4.13.

Let M be the set-incidence matrix where $M_{ij} = 1$ if j is in set i. We start with an integer linear program based on the set-incidence matrix M. Let m_i be the i-th row of M. Let x_i be the decision variable that is 1 if the i-th element is included in the set cover, and 0 otherwise. The ILP is

$$\begin{array}{rcll} \min & \sum x_i & & \\ m_i x & \geq & 1 & (\forall i) \\ x & \in & \{0,1\} & \end{array}$$

Now we solve the LP relaxation; this gives fractional x_i such that $m_i x \geq 1$ but $\sum x_i$ is at most the optimum set cover. Now let us multiply each x_i by $3 \ln n$. If this creates any $x_j \geq 1$, we include j in our set. For all $x_j < 1$, we include x_j in our set with probability x_j. Now consider some m_i. There are two possibilities. If the scaled up $x_j \geq 1$ for some $m_{ij} = 1$, then we take x_j and thus cover set S_i. In the case that all the x_j for which $m_{ij} = 1$ are less than 1, then we have a set of Poisson trials to decide which of them get included in our set. But note that the expected number of elements we include is

$$\sum_{j \in S_i} x_j \geq 3 \ln n$$

since originally the sum exceeded one and we multiplied by $3 \ln n$. Now a Chernoff bound argument shows that we include at least one element with probability at least $1 - 1/n^{3/2}$. So with high probability we include an element from every set.

It remains to analyze the value of the solution. Its expected value is at most $3 \ln n$ times the optimum, so by applying a Chernoff bound, we see that we get within $6 \ln n$ of the optimum with probability $> 1 - 1/n$.

We can combine the high probability results to say that with high probability we get a solution that is both feasible and cheap (this requires saying; a priori one might fear that if we condition on being feasible, the cost goes up too high). If we don't we can keep trying until we do (note that we can compare our cost to the cost of the optimal LP solution to get a bound on the quality of our solution); this gives a Las Vegas algorithm.