Randomized Algorithms assignment 6

Rasmus Stavenuiter, Jacob Harder

1 Summary

This summary is intentionally left blank.

Proof of corollary 5.12

By assumption we have that any vertex of any dependency graph can have degree at most d.
Furthermore the assumption that $ep(d+1) \leq 1$ implies

$$p(d+1) \leq \frac{1}{e} \leq \left(1 - \frac{1}{d-1}\right)^d$$

which implies

$$p \leq \frac{1}{d+1}\left(1 - \frac{1}{d-1}\right)^d \leq \frac{1}{d-1}\left(1 - \frac{1}{d-1}\right)^d$$

Hence for $d > 2$ we let $x_i = \frac{1}{d-1}$ for $i = 1, \cdots, n$ and then we obtain for any $i = 1, \cdots, n$ that

$$x_i \prod_{(i,j)\in E} (1 - x_i) \geq \frac{1}{d-1}\left(1 - \frac{1}{d-1}\right)^d \geq p = P(\mathcal{E}_i)$$

so by the Lovasz local lemma we obtain that

$$P\left(\bigcap_{i=1}^{n} \overline{\mathcal{E}_i}\right) \geq \prod_{i=1}^{n} (1 - x_i) = \left(1 - \frac{1}{d-1}\right)^n > 0$$

as desired.

If $d = 1$ we have

$$\left(1 - \frac{1}{d+1}\right)^d = \frac{1}{2} \geq \frac{1}{e}$$

and if $d = 2$ we have

$$\left(1 - \frac{1}{d+1}\right)^d = \left(\frac{2}{3}\right)^2 = \frac{4}{9} \geq \frac{1}{e}$$

so in either case we can apply the same argument with $x_i = \frac{1}{d+1}$.

Problem 5.13

Let $v_1, \cdots, v_n$ denote the vertices of the graph.
As in section 5.6 of Motwani & Raghavan we consider the decision tree of the experiment.
Each node is labelled by two sets $[A, B]$ corresponding to the assignments made so far, i.e. the root r is labelled $[\emptyset, \emptyset]$ and if a is any node at level i labelled by $[A, B]$ we have $\#(A \cup B) = i$, and if c and d are its left and right child respectively, they will be labelled $[A \cup \{v_{i+1}\}, B]$ and $[A, B \cup \{v_{i+1}\}]$ respectively, and the algorithm proceeds equiprobably to either child.

Now for a node, a, in the decision tree let $E(a)$ denote the expected number of crossing edges conditioned on reaching node a. Then clearly

$$E(a) = \frac{E(c) + E(d)}{2}$$

so that $\max\{E(c), E(d)\} \geq E(a)$. Furthermore we have seen in theorem 5.1 that $E(r) \geq m/2$.
Hence it is possible to make choices that do not decrease the expected number of crossing edges all the way from the root to a leaf, and since there is no more randomness at a leaf, l, the number of crossing edges is equal to $E(l)$, and hence making the choices such that the expected number of crossing edges do not decrease will yield a partition where this number is at least $m/2$.

So we are left with the task of determining which of $E(c)$ and $E(d)$ is the larger.
To this end consider a node, a, at level i in the tree, i.e. proceeding to either the left or the right child corresponds to assigning v_{i+1} to A or B respectively. Let k_A denote the number of crossing edges from v_{i+1} to A and define k_B analogously.

Now observe that after assigning v_{i+1} we still have for any edge (u, v), where either or both of it end points have not yet been assigned, that it will become a crossing edge with probability $\frac{1}{2}$. Furthermore, the number of such edges will be the same regardless of the assignment of v_{i+1}; it will be the number of such edges before the assignment of v_{i+1} minus the number of edges one of whose end points is v_{i+1} and the other is in $A \cup B$. Call this number N_0. Then since assigning v_{i+1} to A adds k_B crossing edges and assigning it to B adds k_A crossing edges we obtain that $E(c) = K_0 + k_B + \frac{N_0}{2}$ and $E(d) = K_0 + k_A + \frac{N_0}{2}$, where K_0 is the number of crossing edges from A to B before assigning v_{i+1}. Hence we have reduced the question to determining the larger of k_A and k_B which can be done by just checking the other end point of all of v_{i+1}'s incident edges.

If there are no multiple edges there can be at most $n-1$ of these and so this procedure is polynomial when performed on all n vertices.
If we allow multiple edges we can merge any multiple edges between to vertices into one giving it an integer weight corresponding to the number of edges between the two vertices. This weight is then used in computing k_A and k_B and these computations are essentially equivalent to the case of no multiple edges.

Thus by the above considerations the deterministic algorithm of computing k_A and k_B for each vertex in turn and assigning it to B if k_A is the largest and vice versa will obtain a partition with at least $m/2$ crossing edges in polynomial time as desired.

Problem 7.1

As observed in the proof of theorem 7.1, the task is equivalent to computing $P(Mr = 0 \mid M \neq 0)$.
So assume that M is a non-zero $n \times n$ matrix and sample $r = (r_1, \cdots, r_n)^T$ by sampling each entry independently and uniformly from $\{0, 1\}$.

Now since M is non-zero, at least one of its row, $M_1, \cdots, M_n$ is non-zero. Let M_i denote this row.
Then with $x = (x_1, \cdots, x_n)^T$, $M_i x$ is a multivariate (non-zero) polynomial of total degree 1, and hence by Schwartz-Zippel $P(M_i r = 0) \leq \frac{1}{2}$, since the $r_1, \cdots, r_n$ are sampled independently and uniformly from a subset of $\mathbb{R}$ of size 2.

Hence

$$P\left(Mr=0\right)=P\left(M_1r=0,\cdots,M_nr=0\right)\leq P\left(M_ir=0\right)\leq\frac{1}{2}$$

proving theorem 7.1 using the Schwartz-Zippel theorem.

2 Problem 7.4

Suppose two integer multisets $X=\langle x_1,\ldots,x_n\rangle, Y=\langle y_1,\ldots,y_n\rangle$ are given. We can associate to a multiset (e.g. X) the polynomial

$$p_X(z)=(z-x_1)(z-x_2)\ldots(z-x_n)$$

It is easy to see that identical multisets give rise to the same polynomial, (by commutativity of multiplication). On the other hand if X and Y are not identical, the multiplicity of at least one root must differ (say non-roots have multiplicity zero) so the polynomials will differ. This means we can tell apart the multisets by determining if their polynomials differ. Observe that p_X is a polynomial in 1 variable which has degree n. By the Schwartz-Zippel theorem we can choose a field $\mathbb{F}$ containing $x_1,y_1,\ldots,x_n,y_n$ and a subset $S\subseteq\mathbb{F}$ with $|S|=2n$ such that

$$P(p_X(r)-p_Y(r)=0\mid p_X-p_Y\not\equiv 0)\leq\frac{1}{2}$$

when $r\in S$ is chosen uniformly at random. If we assume multiplication in $\mathbb{F}$ is $O(1)$ we get a Monte Carlos algorithm with 2^{-m} probability of failure running in $O(mn)$ time - an improvement to the $O(n\log n)$ of the sorting algorithm.

We now move on to discuss what happens if we have to consider the sizes of the integers involved and the time complexity of comparison and multiplication of such. Assume the integers in the multisets are all less than k. We can convert them to $\ell=\lceil\log_2(k)\rceil$ bit strings in $O(n)$ time. To determine equality of two of the integers takes $O(\ell)$ time. Thus the sorting algorithm requires $O(n\log n\log k)$ time.

Looking at our randomized algorithm we have a problem with the values of the polynomials becoming very large, in fact at the order k^n requiring us to work with ℓn bit numbers if we use $\mathbb{R}$ as the field. However we could choose a prime $2k>p>k$ and work with the finite field $\mathbb{F}_p$. This garantees $\ell=\lceil\log_2(p)\rceil=O(\log(k))$ bit numbers are enough. Multiplication of ℓ bit numbers is actually $O(\ell^2)=O(\log k)$. So we can evaluate our polynomials in $O(n \log k)$ time. All in all we get $O(nm \log k)$ running time with success probability $1 - 2^{-m}$ still an improvement to $O(n \log n \log k)$.

However if we want to convert from Monte Carlo to Las Vegas: To garantee that two multisets are equal we need to check at least $n+1$ different values from S yielding a bad running time of $O(n^2 \log k)$.