Randomized Algorithms 2024/5
Notes on Homework Set 2

Moni Naor

The simultaneous message model: How to get an $O(k \log k)$ protocol for the problem of intersection of sets of size k where Alice, who gets $S_A \subseteq U$, and Bob, who gets $S_A \subseteq U$, and who share a random string and should send a single message to a referee which will determine whether the intersection $S_A \cap S_B$ is non empty or not. The idea is to map the range into a domain of size roughly $2k^2$ using a hash function defined by the shared randomness.

After each parties compute the hash of it set, it sends the referee the list of hash values, which is at most $k \log(2k^2)$ which is $O(k \log k$ bits long. The referee sees whether there are two hash values that are the same and declare "non empty" if this is the case. Note that a non-empty intersection will be declared as such. We need to analyze what happens in this case.

If the two sets are disjoint, the good event is when no element in S_A has the same hash value as an element in S_B. There are k^2 such potential collisions and the probability for a collision is $1/(2k^2)$. By the union bound the probability that an unwanted collision occurs is at most $1/2$. We need very little from that hash function, just that any two values are the same should be bounded by $1/(2k^2)$, so pair-wise independence suffices.

Stream verifying a proof of Hamiltonian cycle: two possible approaches to this problem are to consider the specific properties of Hamiltonian cycles (e.g. the fact that they include every node exactly twice) or to construct a general solution for all problems in NP. For any language $L \in NP$ we know that there is a witness checking poly-time procedure $W_L \colon \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}$ so that for any $x \in L$ there is a $y \in \{0,1\}^{poly(|x|}$ s.t. $W_L(x,y) = 1$ and for all $x \notin L$ and all $y \in \{0,1\}^*$ we have $W_L(x,y) = 0$. The question is how to turn the procedure into one requiring logarithmic space to get a $1/n$ error,

The idea I had in mind is that we can use the memory-checking mechanism we saw in class to simulate any NP verifier that wants to simulate the procedure W_L to check a witness, but with only a small secret memory. The original procedure for W_L uses a lot of memory, for instance, storing x and then performing all sorts of operations depending on the stored x. But the point is that the procedure is deterministic and the proving attaching a proof to the string can know ahead of time which read it makes and provide the appropriate value, including the timestamp. All the low memory verifier needs to do is execute the steps of the memory checker, to see that it is consistent and of course that the procedure W_L accepts at the end given a consistent memory. The probability of failure is the probability that the memory checker fails.

Homework: show that if a language can be accepted by a low memory verifier (log space) receiving

a one-way proof or streaming, as the one above, then it is in NP. This gives us an alternative definition of NP.

A heavy cut: for a random partition of the nodes into two sets, for each edge the probability that it is in the cut is 1/2. So the expected value of the cut size is $1/2|E|$ and therefore there must be a cut achieving at least the expectation.

How about finding such a cut? The argument needed just pair-wise independence between the assignment of the endpoints, so taking all functions in a family pf pairwise independent functions and trying all the possibilities until finding one that defines a heavy cut will work. A natural family for this purpose is the $h_r(x) = \langle x, r \rangle$ where $\langle x, r \rangle$ is the inner product over $GF[2]$ and r is a random vector. The length of the vector r is $\lceil \log n \rceil$

Another possibility is to add the nodes one by one in a greedy manner, putting each node in the part containing at least half its neighbors previously assigned a part.

References

[1] Cynthia Dwork, Moni Naor, Guy N. Rothblum, Vinod Vaikuntanathan: How Efficient Can Memory Checking Be?. TCC 2009: 503-520

[2] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, Moni Naor: Checking the Correctness of Memories. Algorithmica 12(2/3): 225-244 (1994).

[3] Richard J. Lipton, Efficient Checking of Computations. STACS 1990: 207-215