CS 744: Pseudorandomness Generators

Lecture 2: Randomness and its benefits

December 31, 2015

We look at a well-studied problem from graph theory: given a graph G and vertices s and t from the graph, we have to determine whether s and t are connected by a path or not.

There is a linear-time algorithm for this, by using either breadth-first search or depth-first search. Both these algorithms also use linear space in the worst case. Is there an algorithm which uses asymptotically lesser space?

There is a famous algorithm by Savitch, which solves reachability in space $\log n$ where n is the number of vertices in the graph. Unfortunately, it uses $n^{\log n}$ time. This recursive algorithm is as follows.

The algorithm has to determine whether there is a path in G from vertex u to vertex v in at most t steps. If t is zero, then the answer is clearly no, unless u is the same as v. If t is one, then the answer is yes if and only if either u is v, or u and v are adjacent to each other. Otherwise, the answer is the same as that obtained by the recursive query of whether there is a vertex w in the graph such that there is a path in G from u to w using $\lfloor t/2 \rfloor$ steps and there is a path in G from w to v using $\lceil t/2 \rceil$ steps.

A node u is reachable in G from v if and only if the above algorithm answers yes with $t = n$. Since the allowed number of steps halve in each call of the algorithm, the call tree of the algorithm has at most $\log n$ levels.

It is possible to do the two recursive queries for each w by using only $\log n$ space to store the w. The key observation is that the first subquery can be executed first, and all that we need to store is the result — one bit – before executing the second subquery to compute the final result. Thus the space required for executing the two subqueries is equal to the maximum space required for a subquery by itself (plus one bit). Thus the space required by the algorithm is $O(\log^2 n)$.

For each recursive call, the algorithm has to iterate over all the possible intermediate vertices w. There are thus $2n$ recursive subqueries per call of the routine, hence the total time taken is $O(n^{\log n})$.[1]

Can we simultaneously reduce the time and the space complexity of the graph-reachability problem?

We show a randomized algorithm for the reachability problem in *undirected* graphs. [2]

The algorithm performs a random walk from s to t of length polynomial in the number of vertices. First, we set the current vertex to the starting vertex. Then, we choose a vertex adjacent to the current vertex uniformly at random and set it as our new "current vertex". Iterate this

[1] Obtained by solving the recurrence $T(n) = 2nT(n/2) + c$.

[2] we will indicate later why undirectedness helps, even though it is not immediately clear. If time permits, we will go into an analysis of the directed case.

process for a number of times polynomial in n. If during this process, we arrive at t, then we say that s and t are connected, otherwise we say that they are not.

The algorithm is fairly simple, but we now have to argue that if s and t are indeed connected, then it will say so with very high probability.

1 Linear Algebra: Mixing Times of Random Walks on undirected graphs