NL-completeness, NL= coNL

Evgenij Thorstensen

V18

Logspace, recap

Defined on machines that have an read-only input tape.

In log space we can store a fixed number of pointers into the input.

Can decide problems decidable by looking at constant-sized pieces of input.

This includes counting, but only up to n.

Logspace, properties

$$\mathsf{L} \subseteq \mathsf{NL} = \mathsf{coNL} \subseteq \mathsf{P}$$

L is low for itself — closed under composition and subroutines.

We have a notion of completeness under logspace reductions.

PATH is NL-complete

We will reduce NTMs to graphs using log space.

Same idea as before, we reduce the NTM deciding some NL language instead of the language itself.

General argument, but logspace bound of NTM crucial to keep logspace bound on reduction.

NTM to PATH

Let M be an NTM for a language in NL, and let w be a string.

The configurations of M on w will be vertices.

Edge from a to b whenever configuration b follows configuration a.

Unique accept configuration (same as for PSPACE).

Accept config reachable from start config if and only if M accepts w.

Let's transduce

A configuration of M on w has size $c \log n$.

Can loop through all $2^{c \log n}$ of them (we have enough time)

Likewise, for edges, loop through all pairs (c_1, c_2), checking whether there is a transition between them for w.

Need to store c_1, c_2, and (constant) change.

Final touches

Logspace reduction bound by construction.

If M accepts w, there is a branch starting at start config and ending at accepting config.

This is case if and only if our graph (of size $2^{c \log n} = n^c!$) has a path between corresponding nodes.

Note the input size blowup, and recall trading time for space trick.

Consequences

Since PATH is solvable in polynomial time by DFS or BFS, $\mathsf{NL} \subseteq \mathsf{P}$.

Since log space reductions are logspace bounded DTMs, they are bounded by polynomial time.

Thus they are restricted polynomial time reductions, and $\mathsf{NL} \subseteq \mathsf{P}$ follows.

coNL

coNL is the class of languages whose complements are in NL.

Since PATH is NL-complete, the complement is coNL-complete.

To show that NL = coNL, suffices to show that NOPATH is in NL.

This result is the Immerman-Szelepcsényi theorem, from 1987.

Proof outline

We will build an NTM that can count the number of nodes reachable from s in a given graph using log space.

Using this, we can solve NOPATH (how?)

Proof outline

We will build an NTM that can count the number of nodes reachable from s in a given graph using log space.

Using this, we can solve NOPATH (how?)

Well, count, then add the edge (s, t), and count again.

We will make this idea more precise.

NOPATH with reachable nodes

Assume we have G, s, t, and c, the number of nodes reachable from s.

We loop over the nodes of G, and for each, we guess if it is reachable. If we guess t, we reject.

If we guessed that it was reachable, we verify this guess by guessing (one node at a time) a path.

If this path fails, we reject.

For each node verified to be reachable, increment a counter. At the end, check whether c equals the counter. If yes, accept.

NOPATH with reachable nodes, reword

We guess a subset of nodes (except t) we think are reachable (one at a time).

We verify each of them by guessing a path (one node at a time).

We branch on every path, and if the path succeeds, we increment a counter.

We accept if we end up with the counter equal to c after getting through all nodes.

Imporant not to *at any point* assume we are returned a yes/no for a path (because nondeterminism does not work like that).

Counting

So, if we knew how many nodes are reachable, we could solve NOPATH.

In fact, Immerman proves that counting gives the complement in general.

So, how do we count reachable nodes?

Need NTM where a branch ends up with correct value of c, and all other branches reject.

Can then glue my machines together.

But didn't we just count reachable nodes already?

Previously described machine sure seems to end up with correct number on accepting branch.

Can't we simply remove c?

But didn't we just count reachable nodes already?

Previously described machine sure seems to end up with correct number on accepting branch.

Can't we simply remove c?

Halting issues

Previous machine does count number of reachable nodes.

But, without c, we would have halting problems.

On some branches, I guessed a subset of the reachable nodes. I verified them all. How do I know not to accept here, too?

If I do all nodes on same branch, on the other hand, how do I deal with non-reachable nodes?

Inductive algorithm

We will calculate reachable number by computing number of nodes c_i reachable in $1 \leqslant i \leqslant m$ edges, using previous value.

Using c_i, can halt if $c_{i+1} = c_i$.

Basically the same machine.

We start with $c_1 = 1$.

Counting

Assume we know c_i. We loop over the nodes of G. For each node v, we guess a set of nodes we believe to be reachable in this many steps (A_i), one at a time.

To verify our guess, we guess a path from s to a_i, and we count successes.

If successful, look in edge table for edge (a_i, v). If exists, increment c_{i+1}.

Need to make sure we didn't end up with a subset of A_i, since we re-guess it all the time.

Aha: Can check if the count for my guessed A_i is correct (using c_i).

Counting, structure

We store two counters, and re-guess the set of nodes previously reached all the time.

Need to, since we can't store it.

Branches that guess subsets/unreachable sets reject.

The non-rejecting branch will end up with correct number of reachable nodes.

At this point, I glue all the branches of my previously described NTM onto here.

NL=coNL, summary

We can count in NL by using counters to avoid subsets of what we need.

Today the technique seems obvious — trade time for space, use counters.

Before theorem was published, many believed NL $\neq$ coNL, by analogy to P and NP.