CSC 2401, Introduction to computational complexity
Lecture #18, Tuesday, November 13, 2001
Lecture: Allan Borodin
Scribe: Yubin Huang

1 Size of Boolean Circuit

1.1 SIZE vs. DTIME

Last time, we discussed the relationship between the time of the uniform TM and the size of the non-uniform circuit. We defined $L \in SIZE(f(n))$ if for all but finite many n, $L_n = L \cap \{0,1\}^n$ is computed by circuits of size no more than $f(n)$. If $t(n)$ is a time constructible function, we have $L \in DTIME(t(n)) \Rightarrow \exists C, \forall n \cdot L_n \in SIZE(C \cdot t(n) \cdot \log t(n))$ where $L_n = L \cap \{0,1\}^n$. Here, we will learn new complexity classes and show more results between these two models.

Theorem 1.1.1: If L is in sufficiently uniform $SIZE(t)$, then L is also in $DTIME(t \cdot \log t)$.

1.2 SIZE vs. SPACE

We defined width at level i as the number of edges from (non-inout) gates at a level $j < i$ to a gate at level i and the width of a circuit as the maximum width at any level. SPACE corresponds to simultaneous width-sIZE. We have $L \in SPACE(w) \sim L \in WIDTH - SIZE(O(w), SIZE(2^{O(w)})$.

2 Boolean Branching Programs

A much better and more widely studied notion of non-uniform space is in terms of Boolean Branching Programs. Boolean Branching Programs are DAGS in which every non-terminal node is labelled by an input variable (say, x_i) and has outdegree 2 (with one edge labelled $x_i = 0$ and the other labelled $x_i = 1$). Terminal nodes (with outdegree 0) are labelled YES (ACCEPT) or NO (REJECT). A branching program computes a Boolean function by following the path determined by the values of the inputs. We discuss two important measures of the Boolean Branching Program, height and size.

(Unfortunatley, we use size as a circuit measure and as a branching program measure and these really measure very different things as we shall see; namely, circuit size relates to sequential time and (the logarithm of) branching program size corresponds to space.

2.1 Height of Boolean Branching Program

The first measure of a Boolean Branching Program (BP) is its *height*. It is clear that every language $L_n = L \cap \{0,1\}^n$ with n variable is computable by a BP with at most height n because the value can be completely determined by probing all the inputs. Height can be thought of as TIME but it becomes more meaningful when we restrict the BP size.

2.2 Size of Boolean Branching Program

Another measure of a Boolean Branching Program is its *size*, which is the number of nodes in the underlying DAG. Thinking of every node as represnting a configuration in some computation, it is reasonable to define log-SIZE of a BP to be SPACE. In particular, a polynomial size Boolean Branching Program $\sim$ non-uniform DLOG. Branching programs provide an intrinsic model for studying space and time-space tradeoffs.

3 Depth of Boolean Circuit

The second most important circuit complexity measure is circuit *depth*. (Some people might insist that circuit depth is the most important measure.) The *depth* of a circuit is the length of a longest path in the circuit from a variable to the output gate. Circuit depth corresponds to non-uniform "parallel time".

By converting a circuit to a tree by copying nodes, every depth-d circuit can be trivially converted to a formula of size $O(2^d)$.

Further, $\exists c$, such that every formula of size t can be computed to circuit (formula) of depth $c \cdot \log t$. In particular, $O(\log n)$ depth corresponds to polynomial formula size. Let l represent the formula size of some Boolean function. Notice that, to evaluate this formula , we can:

1) Find some node defining a subtree formula size (i.e. the number of leaves in the subtree) l' where $\frac{l}{3} < l' \leq \frac{2}{3}l$;

2) Create a new variable z representing the result of this sub-circuit;

3) Then given the value of z, we can evaluate the original formula with constant additional depth. This results in the recurrence: $DEPTH(l) \leq DEPTH(\frac{2}{3}l) + O(1)$.

Then we can draw the conclusion above.

3.1 Some results

We have following theorems.

Theorem 3.1.1: $\forall n \cdot L_n \in DEPTH(n - \log\log n + O(1))$.

Theorem 3.1.2: $\exists L\ \forall n \cdot L_n$ requires depth $\geq n - \log\log n - O(1)$

That is, the maximum depth required for any L_n is pretty well understood and one can show that "most" L_n require close to this maximum depth. But as in the case of size we do not have natural problems (e.g. in NP) which *provably* require anything close to this bound. Indeed we cannot prove depth bounds $\geq (1+\epsilon)\log(n)$ for any $\epsilon > 0$.

3.2 Classical circuit complexity question

Question: Can every language $L_n \in SIZE(t)$ be computed by a circuit of depth $c \cdot \log t$ or $c_1 \cdot (\log t)^{c_2}$?

4 Depth vs Space

Theorem 4.1: $L \in NSPACE(s) \Rightarrow \exists c \cdot L_n \in (uniform)DEPTH(c \cdot s^2)$.

Theorem 4.2: $L_n \in (uniform)DEPTH(d) \Rightarrow L \in DSPACE(d)$.