CSE 200 - Computability and Complexity
Circuits: Non-uniform vs. uniform

Russell Impagliazzo and Jiawei Gao

Oct., 2016
Yet another model of computation

Today, we’ll introduce *Boolean circuits*, yet another model of computation. In some sense, Boolean circuits are the epitome of breaking down computation into a sequence of finite steps, and are in this sense the most limited model we will consider. On the other hand, the circuit model is *non-uniform*, in that the “algorithm” for one size input needn’t have anything to do with the algorithm for other sizes, which makes it more powerful than other models, and maybe unrealistically powerful.

Next class, we will use results about Boolean circuits to begin our exploration of *NP – complete* problems.
What is a Boolean circuit?

A Boolean circuit takes $n$ Boolean inputs $x_1, \ldots, x_n$, and computes a series of Boolean operations on them. We can view the circuit as a series of gates, $g_1 \ldots g_m$, saying which operations are computed when.

The first $n$ gates are inputs, $g_1 = x_1, \ldots, g_n = x_n$.

Each intermediate gate, $g_i$, $i = n + 1 \ldots m$, computes a given operation $op_i(u, v)$ mapping two bits to one bit, e.g., $u \land v, u \lor v, \neg u$, etc., on two previously computing gates, $g_j$ and $g_k$. In other words, $g_i$ is defined by $g_i = op_i(g_j, g_k)$ for $j, k < i$ (computing functions of the input is a special case.) The output gate is the last gate, $g_m$. (For a binary output).

(Electrical engineers also allow feedback in their circuits, but here we don’t, making the structure of the circuit a DAG.)
Example

For example, say we want to compute whether at least two of the three variables $x_1, x_2, x_3$ are 1. We can do by the following series of gates:

1. $g_1 = x_1$
2. $g_2 = x_2$
3. $g_3 = x_3$
4. $g_4 = g_1 \land g_2$
5. $g_5 = g_1 \lor g_2$
6. $g_6 = g_5 \land g_3$
7. $g_7 = g_4 \lor g_6$
Circuit size function

Let $L$ be a decision problem over a binary alphabet. Then we can let $L_n$ be $L$ restricted to inputs of size $n$. We let $S(L_n)$ be the minimum number of gates in a circuit that computes membership in $L_n$. We say $L \in P/poly$ (the name is from Karp-Lipton, but I’m not going to explain it) if $S(L_n) \in O(n^k)$ for some $k$. 
Note that the above definition has no restriction that circuits for inputs of size $n$ look anything like those for other sizes. This means that $L \in P/poly$ could even be undecidable. For example, let $L = \{x \mid |x| \in Halt\}$, those inputs where their length as a binary string codes a halting machine and input. While $L$ is still undecidable, for each particular length $n$ either $L_n$ is all strings of length $n$ or completely empty. Thus, $S(L_n) \leq n + 1$, and $L \in P/poly$. 


Bounds on worst-case size

The circuit complexity of any function, regardless of complexity, is easily seen to be at most $O(n2^n)$ by explicitly writing out each entry in the truth table. Riordan and Shannon improved this to $O(2^n/n)$ and showed most functions require complexity $\Omega(2^n/n)$, so this is the tight worst-case. Stockmeyer and Meyer showed a similar lower bound for an explicit function, true statements in weak monadic second order theory of a single successor (WSIS), which is decidable but not in any fixed number of exponentials. Unlike TM’s this gives specific numerical bounds, e.g., $S(\text{WSIS}_{610}) > 10^{125}$. On the other hand, the previous best lower bound for any function in $P$ was just improved last year: from Blum’s 3$n$ lower bound in 1984 to a spectacular new lower bound of 3.011$n$ by Find Golovnev Hirsch and Kulikov.

A major challenge in complexity is to find lower complexity examples of functions that are exponentially hard for circuits (or even super-linear).
On the other hand, we could insist that the circuits make sense, and be easy to compute themselves. We say a circuit family, $C_1, \ldots, C_n, \ldots$ is $P$-uniform if given $n$, we can construct $C_n$ in time $\text{poly}(n)$. (Unfortunately, to say this we need to refer to a non-circuit model of computation, such as TM.) We will show today: There is a $P$-uniform circuit family computing each $L_n$ if and only if $L \in P$. In this weak sense, circuits are equivalent to the other models of computation we’ve seen.
The easy direction

Say there is a \( P \)-uniform circuit family \( C_n \) computing \( L_n \). To give a poly-time algorithm to decide whether \( x \in L \), we

1. Let \( n = |x| \)
2. Compute \( C_n \) using the uniformity
3. Evaluate \( C_n \) gate by gate, using the defining equation
4. Accept if the last gate is 1, reject if 0.
Simulating a TM with a circuit

We need to show that whatever is solvable in polynomial time has a uniform circuit family computing it of polynomial size. So we need to give a recipe for translating TM algorithms to Boolean circuits. For simplicity, we’ll simulate a one-tape TM, but the method easily generalizes (see class notes). Say $M$ is a one-tape $TM$ that decides $L$ in polynomially bounded time $T(n)$. $M$ can also use at most $T(n)$ cells of memory.
Coding configurations as Boolean variables

To simulate $M$, at each time step $t = 1 ... T$, we need to keep track of the configuration of $M$:

1. The state of $M$, $q_t$
2. The position of the tape head at time $t$
3. For each of the first $T$ cells of the tape, the content of the cell
Coding configurations as Boolean variables

We’ll use Boolean indicator variables, each of which will be a gate in the circuit.

1. \( \text{State}_{t,q} = 1 \) iff the state at time \( t \) is \( q \)
2. \( \text{Position}_{t,i} = 1 \) iff the tape head at time \( t \) is at \( i \)
3. \( \text{Cell}_{i,t,\sigma} = 1 \) iff the symbol at cell \( i \) at time \( t \) is \( \sigma \).
Initialization

To describe the first configuration, $t = 1$, all of the variables are either constants or input bits or their negations. For example, $State_{1,q_0} = 1$; $Cell_{i,1,1} = x_i$ for $i = 1..n$, $Cell_{i,1,0} = \neg x_i$ for $i = 1..n$. 
Simulating one step

To see what action to take in step $t$, we need to know what the current state is (have that) and what the symbol under the tape head is.

We can express “The tape head is reading $\sigma$” as:

$$read_{t,\sigma} = \bigvee_{i=1}^{T} (Position_{t,i} \land Cell_{i,t,\sigma}).$$

Here, we are using unbounded fan-in or as shorthand for a sequence of or operations.
Simulating one step

Now that we have the read symbol coded, we can figure out what actions to take, and what state is next.

We can define

\[ State_{t+1,q} = \bigvee_{q',\sigma: \delta(q',\sigma) = (q,action)} (State_{t,q'} \land read_{t,\sigma}). \]

Similarly, for each type of action (left, right, stay, accept, write(\sigma), etc.) We can define

\[ action_{t+1} = \bigvee_{q',\sigma: \delta(q',\sigma) = (q,action)} (State_{t,q'} \land read_{t,\sigma}). \]
Performing actions

1. $Cell_{i,t+1,\sigma} =$

2. $Position_{i,t+1} =$
Output

\[ \text{Accept} = \bigvee_t \text{accept}_t. \]
Let’s go back and look at the total size of the circuit we created. We have $O(T^2)$ Cell variables, and the other types are $O(T)$. The biggest expression is for *read*, the new state and actions, which are $O(T)$ long; the others are constant size. Since there are only $O(T)$ variables with $O(T)$ size expressions, and $O(T^2)$ variables with constant size expressions, both contribute $O(T^2)$ to the size.
It should be pretty clear that we can automate this conversion process. All steps are purely syntactic; most complex operations required are keeping a counter for $t$. 
Circuit SAT

Our first $NP$-complete problem will involve circuits. Let $C(x_1, \ldots x_n)$ be a circuit with $n$ variables. We say values $x_1 = a_1, \ldots x_n = a_n$ is an assignment, and it satisfies $C$ if $C(a_1, \ldots a_n) = 1$. The Circuit-SAT problem is to decide whether a satisfying assignment exists.
Special cases of interest

A \textit{CNF} is a very special form of circuit. A literal is a variable or its negation. A clause is an OR of literals. A conjunctive normal form circuit is an AND of clauses.
The width of a clause is the number of literals in it. Every function can be written as a CNF, but functions such as parity require all variables in some clause, so the width might be as large as $n$. $k$-CNF circuits are CNF’s with every clause having width at most $k$. This means they only compute some very special types of functions. Nevertheless, we’ll show that $k$ – SAT, the satisfiability problem restricted to $k$-CNF’s, is also NP-complete.