CIRCUIT COMPLEXITY: ONE POPULAR APPROACH TO P VERSUS NP

JOEL FRIEDMAN

Contents

Disclaimer: The material may sketchy and/or contain errors, which I will elaborate upon and/or correct in class. For those not in CPSC 421/501: use this material at your own risk...

The standard way of trying to solve P versus NP is indicated in Section 9.3 of the textbook [Sip]. To understand the idea, let us recall some facts about Boolean formulas and circuits, and about our proof of the Cook-Levin theorem.

1. The "Formula Size Complexity" of a Boolean Function

If f is a Boolean formula on variables $x_1, \ldots, x_n$, then the *size* of f is the number of variables in the formula, e.g.,

$$\mathrm{Size}\Big((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)\Big) = 4.$$

A formula can be viewed as a tree, whose leaves represent literals and whose interior nodes represent subformulas; for example, the above formula as a tree is: In this way the root of the tree represents the overall formula, and the leaves of the tree are the literals, i.e., $x_1, \neg x_1, \ldots, x_n, \neg x_n$.

In a Boolean formula, we can use DeMorgan's laws to assume that all negations occur at the leaves, e.g.,

$$\neg\Big((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)\Big) = \big(\neg(x_1 \wedge \neg x_2)\big) \wedge \big(\neg(\neg x_1 \wedge x_2)\big)$$

$$= (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2).$$

Research supported in part by an NSERC grant.

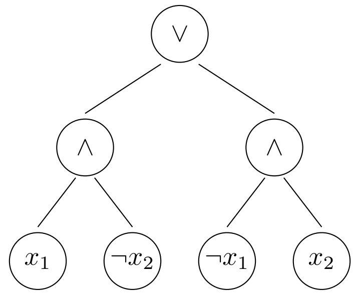

FIGURE 1. $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ as a tree.

Hence, allowing any literal (i.e., and x_i or $\neg x_i$ for some i) on the leaves, the size of a formula is the number of interior nodes plus one, and each interior node is either the AND or OR of its two children.

By a *Boolean function of n variables* we mean a map $f = f(x_1, \dots, x_n)$ from $\{T, F\}^n \to \{T, F\}$. By the *formula size complexity* of f (also called the *minimum formula size* of f) we mean the minimum size formula needed to compute f. Intuitively this is one measure of how "complex" Boolean function is. Assuming our formulas only use $\neg, \vee, \wedge$, we have

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

(where $\oplus$ is the exclusive-or, which in [Sip] is called the *parity* function of x_1, x_2); it is not hard to see that this is the smallest formula that represents $x_1 \oplus x_2$, and hence

$$\mathrm{MinFormulaSize}(x_1 \oplus x_2) = \mathrm{FormulaSizeComplexity}(x_1 \oplus x_2) = 4.$$

It is a classic problem in computer science to determine the formula size complexity of various Boolean functions. On the homework we have seen that any Boolean function on n-variables can be expressed in a Boolean formula of size at most $n2^n$. The number of Boolean functions on n-variables is the number of maps $\{T, F\}^n \to \{T, F\}$, which is 2^{2^n}. One can easily show that most Boolean formulas on n variables—say over 99% of them—require a formula of size at least $2^n/(3n)$ for large n, simply by showing that the total number of Boolean formulas of size $2^n/(3n)$ or less is significantly less than 2^{2^n} (below we outline this computation).

Hence average size of a Boolean function on n variables is somewhere between roughly $2^n/n$ and $n2^n$. As of today, the best lower bound on the formula size for a reasonably "explicit" Boolean function is roughly n^3. Hence there is a huge gap between the minimum formula size that one can prove for an explicit function (as of today—a number of interesting advances have occurred recently) and the typical formula size.

2. Circuit Size Complexity of a Boolean Function

A related concept to a Boolean formula is a Boolean circuit. The idea is explained on pages 380,381 of [Sip] and illustrated in a number of figures there. Formally a circuit on n Boolean variables $x_1, \dots, x_n$ is a sequence of variables $y_1, y_2, \dots, y_m$ where each y_i is built from one or two of the "previous" variables, $x_1, \neg x_1, \dots, x_n, \neg x_n, y_1, \dots, y_{i-1}$, either as the negation of one previous variable or the AND or OR of two previous variables. We refer to the $y_1, \ldots, y_m$ as the *gates* or *interior nodes* of the circuit.

A Boolean formula is a special type of Boolean circuit, where each y_i can only be used once by the variables that follow, $y_{i+1}, \ldots, y_m$ (in a circuit, each y_i can be used any number of times). See Figures 9.23, 9.24, and 9.26 of [Sip] for examples of circuits; circuits can be viewed as *directed acyclic graphs*, just as a formula can be viewed as a tree. By contrast, in Figure 1 above, each interior node is the child of only one node. We define the *size* of a circuit to be the number, m (of variables $y_1, \ldots, y_m$ involved in the computation).

One defines the *circuit size complexity* of a Boolean function to be the minimum size of a circuit that expresses a Boolean function. Since each formula gives rise to a circuit, we have

$$\mathrm{CircuitSizeComplexity}(f) \le \mathrm{FormulaSizeComplexity}(f)$$

for any Boolean function, f.

Again, simply by counting the number of circuits there are of size s or less in n variables, and realizing that the number of Boolean functions of n variables is 2^{2^n}, we can show that most Boolean functions have complexity at least $2^n/(3n)$.

3. The Boolean Functions 3COLOUR

If $m \in \mathbb{Z}$, a graph with vertex set $[m] = \{1, \ldots, m\}$ can be described by $n = \binom{m}{2} = m(m-1)/2$ Boolean variables, x_{ij}, where $1 \le i < j \le m$ and x_{ij} is true when the graph contains the edge $\{i, j\}$. We can therefore define for any $n = \binom{m}{2}$ the function $f_{3\mathrm{COL},n} = f_n(x_{12}, x_{13}, \ldots, x_{m-1\,m})$ that is true if the graph represented by the x_{ij} is 3-colourable.

We now explain that if you can prove that

$$\mathrm{CircuitSizeComplexity}(f_{3\mathrm{COL},n}) \ge n^c$$

for any fixed $c \in \mathbb{N}$, then $\mathrm{P} \ne \mathrm{NP}$.

4. The Cook-Levin Theorem and Deterministic Turing Machines

The Cook-Levin theorem was proven by taking a triple (M, w, N) where M is a non-deterministic Turning machine, $w \in \Sigma^*$ is an input to M, and $N \in \mathbb{N}$, and producing a formula that is satisfiable iff M accepts w in N steps. Here N is an arbitrary number, but for the Cook-Levin theorem we assume that N is a polynomial in $|w|$, since this is the case for languages in NP. Recall that the formula was based on Boolean variables

$$\begin{aligned} x_{ijk} = T &\quad \text{iff at time } i\text{, cell } j \text{ contains symbol } k \\ y_{ij} = T &\quad \text{iff at time } i\text{, the tape head is over cell } j \\ z_{is} = T &\quad \text{iff at time } i\text{, the computation is in state } s \end{aligned}$$

If the computation takes time N, then i ranges over $0, 1, \ldots, N$, j ranges over $1, \ldots, N+1$, k over the number of tape symbols and s over the number of states. The number of variables is therefore $O(N^2)$. The size of the formula produced was also $O(N^2)$.

(The textbook [Sip] uses slightly different Boolean variables based on the way it denotes the configuration of a Turing machine.)

Now consider what happens when M is a **deterministic** Turing machine. In this case, for each i, the time i variables, i.e., x_{ijk}, y_{ij}, z_{is} are deterministic functions of the time $i-1$ variables (i.e., of $x_{i-1\,jk}, y_{i-1\,j}, z_{i-1\,s}$ where j,k,s vary over all possible values). In this way for fixed i,j,k,

$$x_{ijk} = \text{some Boolean function of } x_{i-1\,jk}, y_{i-1\,j}, z_{i-1\,1}, \ldots, z_{i-1\,|Q|},$$

and similarly for the y_{ij}, z_{is}. Since the Boolean functions to compute the time i variables in terms of the time $i-1$ variables are of a bounded number of variables, we get a circuit of size $O(N^2)$ to compute z_{Ns} as a function of the input, which tells us if M accepts w in N steps.

Hence, for example, if 3COLOUR is computable by an $O(n^k)$ time deterministic algorithm, there are $O(n^{2k})$ size circuits to compute whether or not a graph on vertex set $[m]$, with $n = \binom{m}{2}$, is 3-colourable. In this case, if f_n denotes the Boolean function for 3COLOUR (described in the last subsection), then

$$\text{CircuitSizeComplexity}(f_n) = O(n^{2k}),$$

and, moreover, the circuits above have a "uniform structure" in the way they work. Hence if you can prove that

$$\text{CircuitSizeComplexity}(f_n) \geq n^c$$

for any fixed c, and n sufficiently large, then you have proven that P $\neq$ NP. On the other hand, if you can prove that

$$\text{CircuitSizeComplexity}(f_n) = O(n^c)$$

for some c, and the circuits you use to compute f_n have a sort of "uniform structure" (we leave this vague), then 3COLOUR $\in$ P and hence P $=$ NP. It is conceivable the above bound holds but that the circuits change "wildly" (this is very vague) for different values of n, in which case you can't tell whether or not 3COLOUR $\in$ P (but this bound would still be a fabulous result, and unexpected by most researchers today).

At present, the only lower bound we know for the circuit size of a "reasonably explicit" Boolean function on n-variables if of size Cn where C is somewhere between 4 and 10 (this C tends to slowly increase over the years). Note that it is clear that any function of n variables that genuinely depends on all its variables must have size at least n (since y_{n-1} can only involve at most $n-1$ of the literals), so this lower bound involves only a constant factor (which tends to require a lot of work) over the trivial lower bound of n (for any function depending on all of its variables).

Exercises

(1) Consider the number, $g(n,m)$, of circuits of size at most m on n variables, where $x_1, \neg x_1, \ldots, x_n, \neg x_n$ are the inputs to (or literals of) the circuit, and $y_1, \ldots, y_m$ are the gates (or interior nodes) of the circuit.
   (a) For each i, show that there are $O(i+n)^2$ choices for how y_i is a function of the literals and $y_1, \ldots, y_{i-1}$.
   (b) Argue (very crudely) that

$$g(n,m) \leq (Cm^2)^m = C^m m^{2m}$$

   for some absolute constant C, provided that $m \geq n$.

(c) Show that for $m = 2^n/(3n)$,

$$\log_2(g(n,m)) \leq 2^n(2/3 + o(1))$$

as $n \to \infty$.

(d) Show that $g(n, 2^n/(3n)) = o(2^{2^n})$ as $n \to \infty$.

(2) The *depth* of a formula is the length of the longest path from a leaf (i.e., a literal) to its root (i.e., which computes the full formula); for example, the depth of the formula in Figure 1 is 2 (the longest path has three vertices and two edges, which is a path of length 2). Similarly the *depth* of a circuit is the length of the longest path from a literal to the last node (which computes the result of the circuit). We define the *minimum formula depth* (or *formula depth complexity*) and *minimum circuit depth* (or *circuit depth complexity*) of a Boolean function in the analogous way that we did for formula/circuit size.

(a) Explain why for any Boolean function, f,

$$\mathrm{MinCircuitDepth}(f) = \mathrm{MinFormulaDepth}(f).$$

(b) Explain why

$$\mathrm{MinFormulaDepth}(f) \geq \log_2\bigl(\mathrm{MinFormulaSize}(f)\bigr).$$

(3) Show that there is a constant C such that

$$\log_2\bigl(\mathrm{MinFormulaSize}(f)\bigr) \leq C\,\mathrm{MinFormulaDepth}(f) \tag{1}$$

for all Boolean functions, f (i.e., C is independent of the number of variables in n). Do this in the following steps.

(a) Show that for any binary tree with n leaves, there is (at least one) interior node with between $n/3 - 1$ and $2n/3$ descendants.

(b) Show that if v is any interior node of a tree that represents a formula f, and if v represents the subformula g, then we may write

$$f = (g \wedge h_F) \vee (\neg g \wedge h_T),$$

where h_F is the formula for f where v is given the value F (false) and the descendants of v are discarded, and similarly for h_T.

(c) Conclude from the above two parts that if

$$D(n) \stackrel{\text{def}}{=} \max\{\mathrm{MinFormulaDepth}(f) \mid \mathrm{MinFormulaSize}(f) \leq n\},$$

then

$$D(n) \leq \max_{n/3-1 \leq k \leq 2n/3} 2\bigl(D(k) + D(n-k-1)\bigr) \leq 4D(2n/3).$$

(d) Conclude that $D(n) \leq 4(1 + \log_{3/2} n)$ and (1).

(e) Show that the function $f(x) = \log(x) + \log(n-x) = \log(x(n-x))$ is maximized over $0 \leq x \leq n$ at $x = n/2$. Use this to prove the improved bound $D(n) \leq 4(1 + \log_2 n)$.

(4) We say that a Boolean function $f\colon \{T,F\}^n \to \{T,F\}$ is *monotone* if for any $x_1,\ldots,x_n \in \{T,F\}$ and any $i \in [n]$ we have

$$f(x_1,\ldots,x_{i-1},F,x_{i+1},\ldots,x_i) = T \quad \Rightarrow \quad f(x_1,\ldots,x_{i-1},T,x_{i+1},\ldots,x_i) = T$$

We say that a formula or circuit is *monotone* if it involves no negations (only $\wedge,\vee$) and only the literals $x_1,\ldots,x_n$ (and not $\neg x_1,\ldots,\neg x_n$). Show that any monotone Boolean function can be expressed by a monotone Boolean formula.

Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, CANADA.

E-mail address: jf@cs.ubc.ca

URL: http://www.cs.ubc.ca/~jf