INTRODUCTION TO SOLVING P VERSUS NP, AND SUBBOTOVSKAYA'S RESTRICTION METHOD

JOEL FRIEDMAN

Contents

Date: Friday 1st December, 2023, at 10:13.
Research supported in part by an NSERC grant.

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. . .

1. Arithmetic Formulas and Circuits

Formulas and circuits occur in many different contexts; for example, formulas—viewed as trees—are crucial to parsing langauges (including natural languages and programming languages). Here we explain the difference between formulas and circuits by discussing the "arithmetic" situation.

1.1. **Arithmetic Fomulas.**

The arithmetic formula

$$(5+6)\times(7-1) \tag{1}$$

can be viewed as a "computation tree," depicted in Figure 1 This diagram is there-

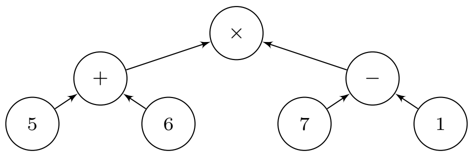


Figure 1. A Simple Arithmetic Formula Tree

fore a *directed graph*, $G = (V, E)$, which is a *tree*, where the vertices of this graph are divided into:

(1) *leaves*, which are vertices with no incoming edges, at the bottom of Figure 1, labelled with constants (in this case $5, 6, 7, 1$, from left to right); and
(2) *internal vertices*, which are vertices with two incoming edges, which are labelled with operations, either $+, -, \times$.

The *root* of this is the vertex at the top, which is the only vertex that has no outgoing edges; every other vertex has exactly one outgoing edge. Each internal vertex can be viewed as computing a function of the leaves, which gives you an algorithm to compute the value at each internal vertex, depicted in Figure 2. The

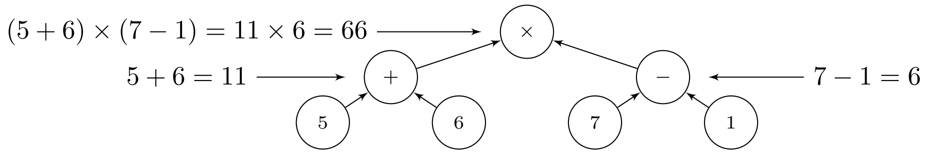


Figure 2. The Values at Interior Vertices

size of a formula refers to the number of leaves, in this case 4; alternatively, one can count the number of operations of the tree, which is 3, always one less than the number of leaves.

Note that using the distributive property of $\times$ over $+, -$, one can write (1) equivalently as

$$(5+6)\times(7-1) = 5\times 7 + 6\times 7 - 5\times 1 - 6\times 1; \tag{2}$$

however, the new formula on the right-hand-side has 7 operations (i.e., the total count of $+, -, \times$), so you'd likely prefer the original formula (1) that requires only 3 operations. Equivalently, the tree corresponding to (2) has 8 leaves, as opposed to the 4 leaves in that of (1).

1.2. **Arithmetic Circuits.** Imagine you want to compute

$$(25+36)^3 \times (12+39)^2 = (25+36) \times (25+36) \times (25+36) \times (12+39) \times (12+39)$$

The formula on the right-hand-size is of size 10 and therefore involves 9 operations (five $+$'s and four $\times$'s); however, you can compute this with only 5 operations (provided that you can "remember" and reuse all intermediate results):

$$y_1 = 25+36,\ y_2 = 12+39,\ y_3 = y_1 \times y_2,\ y_4 = y_1 \times y_3,\ y_5 = y_3 \times y_4.$$

The above is called a *straight line program*, where $25, 36, 12, 39$ are its *inputs*, and the $y_1, \ldots, y_5$ are its *operations* (i.e., each y_i is one of $+, -, \times$ applied to two values, each of which is an input or a y_j with $j < i$). This straight line program can be drawn as a "circuit" depicted in Figure 3. The point is that the interior vertices

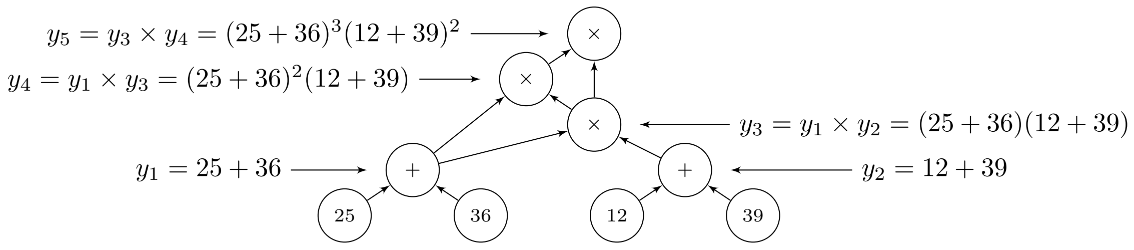

$y_5 = y_3 \times y_4 = (25+36)^3(12+39)^2$
$y_4 = y_1 \times y_3 = (25+36)^2(12+39)$
$y_3 = y_1 \times y_2 = (25+36)(12+39)$
$y_1 = 25+36$
$y_2 = 12+39$
×
×
×
+
+
25
36
12
39

FIGURE 3. An Arithmetic Circuit

representing y_1 and y_3 have more than one arrow leaving them, representing the fact that once we compute y_1, y_3, we can "remember" these values and use them more than once.

In terms of graph theory, a circuit is a *directed graph* that is *acyclic*[1] and has a unique vertex with no outgoing edges (again, called the *root* of the circuit); in addition, for each "input" of the straight line program, the graph has a *source* (i.e., a vertex without incoming edges) representing the input; furthermore every vertex that isn't a source has two incoming edges.

In this way a formula over $+, -, \times$ is a special case of a circuit over $+, -, \times$, where circuits can have more than one outgoing edge from any vertex.

Of course, there is nothing special about the values $25, 36, 12, 39$ in the above circuit, and by replacing these values with variables x_1, x_2, x_3, x_4 we can say that the algorithm

$$y_1 = x_1 + x_2,\ y_2 = x_3 + x_4,\ y_3 = y_1 \times y_2,\ y_4 = y_1 \times y_3,\ y_5 = y_3 \times y_4,$$

gives an algorithm for computing the polynomial

$$y_5 = p(x_1, x_2, x_3, x_4) = (x_1 + x_2)^3 (x_3 + x_4)^2. \tag{3}$$

We may equivalently describe this algorithm by the circuit depicted in Figure 4.

[1] *Acyclic* means that the directed graph has no directed cycle; equivalently, the vertices can arranged as $v_1, \ldots, v_N$ so that all edges have a tail v_i and a head v_j such that $i < j$.

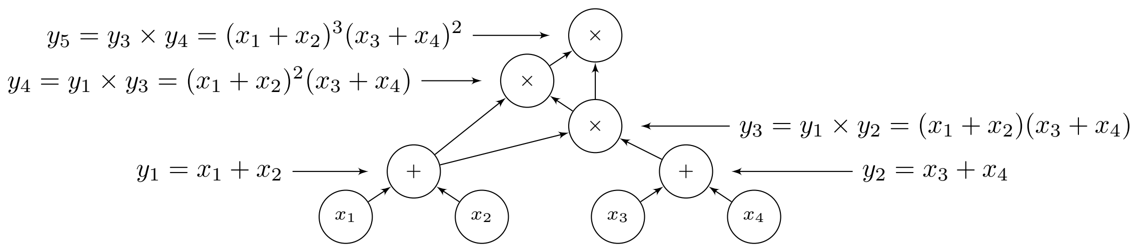

$y_5 = y_3 \times y_4 = (x_1+x_2)^3(x_3+x_4)^2$
$y_4 = y_1 \times y_3 = (x_1+x_2)^2(x_3+x_4)$
$y_3 = y_1 \times y_2 = (x_1+x_2)(x_3+x_4)$
$y_1 = x_1 + x_2$
$y_2 = x_3 + x_4$
$\times$
$+$
x_1
x_2
x_3
x_4

FIGURE 4. An Arithmetic Circuit with Variables at the Leaves

2. Boolean Formulas and the Minimum Size Formula Challenge

Boolean formulas are circuits the natural analog of arithmetic formuals and circuits.

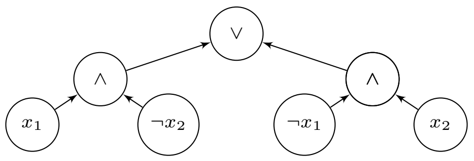
2.1. **Boolean Formulas.**

Recall from our discussion of SAT and 3SAT (see also Section 7.4 of [Sip]), that—informally—a Boolean formula is any formula involving Boolean variables and operations; typically we restrict ourselves to the operations $\wedge, \vee, \neg$.

Just like any arithmetic formula has a corresponding tree, the Boolean formula:

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \tag{4}$$

has a corresponding evaluation tree:



We remark that any Boolean formula using $\wedge, \vee, \neg$ can be written using only $\wedge, \vee$ applied to *literals*, meaning variables and their negations (this was helpful when working with SAT and 3SAT): the way to do this is to use the equalities

$$\neg(y_1 \wedge y_2 \wedge \ldots \wedge y_m) = (\neg y_1) \vee (\neg y_2) \vee \ldots \vee (\neg y_m), \tag{5}$$
$$\neg(y_1 \vee y_2 \vee \ldots \vee y_m) = (\neg y_1) \wedge (\neg y_2) \wedge \ldots \wedge (\neg y_m), \tag{6}$$
$$\tag{7}$$

to "move" all $\neg$ "inside all parenthesis"; for example,

$$\neg\big((x_1 \vee x_2) \wedge (\neg x_3)\big) = \neg(x_1 \vee x_2) \vee \neg(\neg x_3) = (\neg x_1 \wedge \neg x_2) \vee x_3,$$

and the last formula, $(\neg x_1 \wedge \neg x_2) \vee x_3$ involves the operations $\wedge, \vee$ applied to literals. [In a formula we also need parenthesis to indicate how the $\wedge, \vee$ are applied; the tree makes this clear and does not require parentheses.]

Notice that moving the $\neg$ inside all parenthesis does not change the *size* of a formula, i.e., the number of literals involved. We also remark that the size is always one plus the number of $\wedge, \vee$ it involves (ignoring the number of $\neg$ involved).

2.2. **Boolean Functions.** A *Boolean function on n variables* is a function $f\colon \{T,F\}^n \to \{T,F\}$; we typically write $f = f(x_1,\ldots,x_n)$, so that $x_1,\ldots,x_n$ are Boolean variables, i.e., take on the values $\{T,F\}$, and for each $(x_1,\ldots,x_n) \in \{T,F\}^n$, $f(x_1,\ldots,x_n)$ is either T (true) or F (false). Every Boolean formula gives a Boolean function; for example the formula (4) can be viewed as a Boolean function

$$f(x_1,x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2), \tag{8}$$

i.e., the function $f\colon \{T,F\}^2 \to \{T,F\}$ whose values are (or "truth table" is)

$$f(F,F) = F,\ f(F,T) = T,\ f(T,F) = T,\ f(T,T) = F.$$

This function is often called the *exclusive or of x_1 and x_2*, and commonly denoted XOR or $\oplus$.

2.3. **The Minimum Formula Size Challenge.** The **minimum formula size challenge** is to decide for any given Boolean function, $f(x_1,\ldots,x_n)$, what is the minimum size of a formula expressing f. This tends to be very difficult. Often on settles on proving upper and lower bounds on this minimum size that are as close as possible. For reasons we explain later, we are often interested in doing this for functions, f, that arise from problems in P or NP (see Section 3).

For example, for any $k,n \in \mathbb{N}$ consider the *threshold function*

$$\mathrm{Threshold}_{k,n}(x_1,\ldots,x_n) \overset{\mathrm{def}}{=} \begin{cases} T & \text{at least } k \text{ of } x_1,\ldots,x_n \text{ equal } T, \text{ and} \\ F & \text{otherwise.} \end{cases}$$

Then clearly

$$\mathrm{Threshold}_{k,n}(x_1,\ldots,x_n) \overset{\mathrm{def}}{=} \bigvee_{1\le i_1<i_2<\cdots<i_k\le n} \bigl(x_{i_1} \wedge x_{i_2} \wedge \ldots \wedge x_{i_k}\bigr), \tag{9}$$

which expresses this function via a formula of $\binom{n}{k}$ clauses, each with k variables. It turns out that one can usually improve on this (by a lot...).

In class we considered the example:

$$\mathrm{Threshold}_{2,4}(x_1,x_2,x_3,x_4) = (x_1\wedge x_2)\vee(x_1\wedge x_3)\vee(x_1\wedge x_4)\vee(x_2\wedge x_3)\vee(x_2\wedge x_4)\vee(x_3\wedge x_4),$$

which is a formula of size 12. The students proposed two variants: the first is that $\mathrm{Threshold}_{2,4}$ is true iff every group of three variables has at least one true value. Hence

$$\mathrm{Threshold}_{2,4}(x_1,x_2,x_3,x_4) = (x_1\vee x_2\vee x_3)\wedge(x_1\vee x_2\vee x_4)\wedge(x_1\vee x_3\vee x_4)\wedge(x_2\vee x_3\vee x_3);$$

this formula is different, and again has size 12. The second variant is an improvement: we write

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) = x_1 \wedge (x_2 \vee x_3 \vee x_4)$$

and

$$(x_2 \wedge x_3) \vee (x_2 \wedge x_4) = x_2 \wedge (x_3 \vee x_4).$$

Hence we get that $\mathrm{Threshold}_{2,4}(x_1,x_2,x_3,x_4)$ is the disjunction of (i.e., OR of)

$$x_1 \wedge (x_2 \vee x_3 \vee x_4),\ x_2 \wedge (x_3 \vee x_4),\ x_3 \vee x_4,$$

which reduces the formula size to 9. Notice that this strategy can similarly give a formula for $\mathrm{Threshold}_{2,n}$ of size

$$n + (n-1) + \cdots + 2 = \binom{n+1}{2} - 1,$$

which saves roughly a factor of 2 over the original $2\binom{n}{2}$. We then gave a strategy to write $\mathrm{Threshold}_{2,n}$ as a formula of size $n\lceil\log_2 n\rceil$ as follows: first note that

$$\mathrm{Threshold}_{2,4}(x_1,\ldots,x_4)=\Bigl((x_2\vee x_4)\wedge(x_1\vee x_3)\Bigr)\vee\Bigl((x_1\vee x_4)\wedge(x_2\vee x_3)\Bigr)$$

is a formula of size 8. Similarly we can write $\mathrm{Threshold}_{2,8}$ as $c_1\vee c_2\vee c_3$ where

$$c_1=(x_2\vee x_4\vee x_6\vee x_8)\wedge(x_1\vee x_3\vee x_5\vee x_7),\quad c_2=(x_1\vee x_4\vee x_5\vee x_8)\wedge(x_2\vee x_3\vee x_6\vee x_7),$$

$$c_3=(x_1\vee x_2\vee x_3\vee x_8)\wedge(x_4\vee x_5\vee x_6\vee x_7);$$

to prove this, if we write $1,\ldots,8$ in binary, note that c_j divides $x_1,\ldots,x_8$ into two groups with x_i placed according to its j-th bit. Similarly, this writes a formula for $\mathrm{Threshold}_{2,2^\ell}$ of size $\ell 2^\ell$ for any $k\in\mathbb{N}$

If n is not a power of 2, setting $\ell=\lceil\log_2 n\rceil$, we have $2^{\ell-1}<n\le 2^\ell$, and also

$$\mathrm{Threshold}_{2,n}(x_1,\ldots,x_n)=\mathrm{Threshold}_{2,2^\ell}(x_1,\ldots,x_n,F,\ldots,F),$$

which yields a formula of size at most $n\ell=n\lceil\log_2 n\rceil$ for $\mathrm{Threshold}_{2,n}$.

Now, imagine the task of trying to prove that the above strategy for writing $\mathrm{Threshold}_{2,n}$ is optimal, or finding a better method...

2.4. Working over $\{0,1\}$, and Parity and Threshold Functions.

Section 9.3 of [Sip] uses the very common alternate notation that writes 1 for T (true) and 0 for F (false); in this case $\oplus$, i.e., XOR, given by (8), becomes

$$1\oplus 1=0\oplus 0=0,\quad 1\oplus 0=0\oplus 1=1,$$

and hence

$$x_1\oplus x_2=(x_1+x_2)\mod 2. \tag{10}$$

This makes it easy to see that $\oplus$ is an associative operator, and hence $x_1\oplus x_2\oplus\ldots\oplus x_n$ is well defined, and, in fact

$$x_1\oplus x_2\oplus\ldots\oplus x_n=(x_1+x_2+\ldots+x_n)\bmod 2.$$

For this reason we define the parity function as

$$\mathrm{Parity}_n(x_1,\ldots,x_n)=x_1\oplus\ldots\oplus x_n,$$

which is 0 or 1 according to whether or not the number of 1's among $x_1,\ldots,x_n$ is even or odd. Similarly, the threshold functions can be defined more simply over $\{0,1\}$ as

$$\mathrm{Threshold}_{k,n}(x_1,\ldots,x_n)\overset{\mathrm{def}}{=}\begin{cases}T & x_1+\ldots+x_n\ge k,\text{ and}\\ F & \text{otherwise.}\end{cases}$$

2.5. Quadratic Size Formulas for Parity.

For each $n\in\mathbb{Z}$, let $F(n)$ denote the minimum formula size for Parity_n. To build formulas for the parity function we note that

$$x_1\oplus\ldots\oplus x_n=f\oplus g,$$

where

$$f=x_1\oplus\ldots\oplus x_{n/2},\quad g=x_{n/2+1}\oplus\ldots\oplus x_n.$$

Since

$$f\oplus g=(f\wedge\neg g)\vee(\neg f\wedge g), \tag{11}$$

a formula for Parity_n can be obtained from 4 formulas for $\mathrm{Parity}_{n/2}$ (for n even); note that in a formula we have to use a separate tree for each occurrence of f and

g on the right-hand-side of (11). Hence $F(n) \le 4F(n/2)$; it follows that if n is a power of 2, then $F(n) \le n^2$. In general there is a unique $\ell \in \mathbb{N}$ with $2^{\ell-1} < n \le 2^\ell$ (namely $\ell = \lceil \log_2 n \rceil$), and in this case we have

$$\mathrm{Parity}_n(x_1, \ldots, x_n) = \mathrm{Parity}_{2^\ell}(x_1, \ldots, x_n, 0, \ldots, 0),$$

which shows that $F(n) \le (2^\ell)^2 \le 4(2^{\ell-1})^2 \le 4n^2$. Hence there is a formula for Parity_n of size $\le 4n^2$.

2.6. **Minimum Formula Size Challenge: Results up to 2023.**

Any formula for a Boolean function $f = f(x_1, \ldots, x_n)$ that is of size less than n must not involve at least one of its variables $x_1, \ldots, x_n$. Hence any function that depends on all of its variables has minimum formula size at least n. For many years no one gave a significant improvement of this obvious lower bound for an explicitly stated function, f.

In 1961, Bella A. Subbotovskaya [Sub61] (sometimes "Subbotovskaja") gave a rather ingenious argument that shows that Parity_n requires formula of size at least $Cn^{3/2}$ for some $C > 0$. Subbotovskaya's method is often called the method of "restrictions" or "random restrictions," for reasons that we explain below. Using this method, a sequence of works showed that for any $\epsilon > 0$, Parity_n requires formula of size at least $Cn^{2-\epsilon}$; this same method shows that a variant of the parity functions known as *Andre'ev's functions* [And87] require formula size $cn^{3-\epsilon}$; see Section 5 and Appendix A below.

3. Boolean Circuits and P versus NP

The main point of this section is to motivate later sections, which focus on the minimum formula size of Boolean functions. This section is independent of the others.

A Boolean circuit is, roughly speaking, the analog of an arithmetic circuit in Boolean algebra. There is a related "minimum circuit size challenge" for Boolean functions; some special cases of this challenge are equivalent to solving P versus NP. This is the point of Section 9.3 of [Sip]; we explain this here.

3.1. **Boolean Circuits.**

A Boolean circuit is the analog of a Boolean formula where we allow more than one outgoing edge to each vertex, which represents the ability to "remember" the intermediate results of these vertices and to reuse them as many times as we like. The example in [Sip], Example 9.25, page 381 gives a circuit for the parity function of x_1, x_2, x_3, x_4. Notice that this circuit contains three operations $\wedge, \vee, \neg$, so the $\neg$ vertices or "gates" take only one input, not two. Unlike formulas, one cannot move the $\neg$ inside the parentheses without changing the circuit size; however, it is easy to see that doing so will increase the number of $\wedge, \vee$ by at most a factor of 2.[2]

Definition 3.1. We define the *size* of a Boolean circuit to be the total number of $\wedge, \vee$ operations used (i.e., we disregard $\neg$).

In this way a formula is a special kind of circuit, and the circuit size of any formula (viewed as a circuit) is one less than the formula size.

[2] This can be done by adding for each interior vertex that computes a function, g, a "mirror" vertex that computes $\neg g$.

It may be more natural to define the circuit size in terms of the total number of $\wedge, \vee, \neg$ gates used; we claim this won't change the circuit size by much: indeed, we need at most one $\neg$ for each input and each $\wedge, \vee$ gate (after it computes its result); hence a circuit with n inputs and m gates $\wedge, \vee$ needs at most $n+m$ negations. Hence the size changes from m to at most $2m+n$. A similar remark applies if one defines the size as the total number of inputs and gates used.

3.2. **Linear Size Circuit for Parity.** Let $G(n)$ be the minimum circuit size needed to write the n variable parity function $x_1 \oplus \cdots \oplus x_n$. Notice that for any $n', n'' \in \mathbb{Z}$ with $n'+n''=n$, $\mathrm{Parity}_n(x_1,\ldots,x_n)$ equals

$$f \oplus g = (f \wedge \neg g) \vee (\neg f \wedge g),$$

where f, g are the parity functions of, respectively, n; and n'' variables. It follows that for any such n', n'', n we have

$$G(n) \le G(n') + G(n'') + 3$$

(since we count only $\wedge, \vee$, not $\neg$ in $G(n)$). In particular, $G(n) \le G(n-1) + G(1) + 3 = G(n-1) + 3$, and it follows by induction, given that $G(1) = 0$, that $G(n) \le 3n-3$. [Example 9.25 of [Sip] shows that $G(4) \le 9$, which matches this bound.]

3.3. **NP-Complete Languages over $\{T,F\}$ or $\{0,1\}$.** It is easy to take an NP-complete language $L \subset \Sigma^*$, and produce an equivalent NP-complete language over a two-symbol alphabet such as $\{T,F\}$ or $\{0,1\}$: for example, since in class we previously described graphs as a language over:

$$\Sigma = \{0,1,\ldots,9,\#\},$$

and $|\Sigma| = 11$, we can write down a 4-bit binary string to represent each symbol in Σ. This gives a map

$$f\colon \Sigma^n \to \{0,1\}^{4n}$$

for each n. Since 3COLOR is NP-complete, we easily see that

$$\text{3COLOR-IN-BINARY} \overset{\text{def}}{=} \{f(\langle G\rangle) \mid G \text{ is a 3-colourable graph}\}$$

is NP-complete.

We remark that when thinking about graphs and formulas and circuits, one can describe graphs as a language over $\{0,1\}$ where each bit has a more direct meaning regarding the graph: namely, if G is a graph on N vertices, we can describe the edges of G as a subset of

$$\Bigl\{\{1,2\},\{1,3\},\ldots,\{N-1,N\}\Bigr\};$$

we can therefore describe such a graph as a $\{0,1\}$-string $\sigma_{1,2}\sigma_{1,3}\ldots\sigma_{N-1,N}$, with for $i<j$, $\sigma_{i,j}=1$ means that G contains the edge $\{i,j\}$, and $\sigma_{i,j}=0$ means that G doesn't. Using the notation $\langle G\rangle_{\text{intuitive}}$ to denote this description of G, we easily see that

$$\text{3COLOR-INTUITIVE} = \{\langle G\rangle_{\text{intuitive}} \mid G \text{ is 3-colourable}\} \subset \{0,1\}^*$$

is NP-complete, but each symbol in this description of G has a very intuitive meaning.

3.4. **P versus NP as a Problem of Minimum Circuit Size.** [Recall the notation $\mathbb{Z}_{\geq 0} = \{0, 1, 2, \ldots\}$ denotes the non-negative integers.] In Section 9.3 of [Sip], the proof of the Cook-Levin theorem is used to show that P versus NP can be viewed as a problem in minimum circuit size.

First, to any language over $\{T, F\}$ or $\{0, 1\}$ we associate a family of Boolean functions, and vice versa.

Definition 3.2. Let $L \subset \Sigma^*$ where $\Sigma = \{T, F\}$. For each $n \in \mathbb{N}$, we define $\mathrm{BoolFunct}_{L,n} \colon \Sigma^n \to \Sigma$ via

$$\mathrm{BoolFunct}_{L,n}(\sigma_1 \ldots \sigma_n) = \begin{cases} T & \text{if } \sigma_1 \ldots \sigma_n \in L, \text{ and} \\ F & \text{otherwise.} \end{cases}$$

We similarly define $\mathrm{BoolFunct}_{L,n}$ when $\Sigma = \{0, 1\}$, with 1 identified with T, and 0 with F.

[Conversely, there is a reverse procedure: if $f_0, f_1, \ldots$ is a sequence of function $f_n \colon \Sigma^n \to \Sigma$ with $\Sigma = \{T, F\}$, we associate to $\{f_n\}_{n \in \mathbb{Z}_{\geq 0}}$ the language

$$L = \bigcup_{n \in \mathbb{Z}_{\geq 0}} \{w \in \Sigma^n \mid f_n(w) = T\};$$

similarly for $\Sigma = \{0, 1\}$.]

Theorem 3.3. *Let $L \subset \Sigma^*$ where $\Sigma = \{T, F\}$ (or $\{0, 1\}$). If $L \in P$, then for each $n \in \mathbb{Z}_{\geq 0}$ there exists a circuit C_n computing* $\mathrm{BoolFunct}_{L,n}$ *such that the size of C_n at most a polynomial in n.*

Proof. Let us outline the proof: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}$ be a **deterministic** Turing machine that decides L in time Cn^k. Let $\{x_{ij\gamma}, y_{ij}, z_{iq}\}$ be the Boolean variables used in our proof of the Cook-Levin Theorem, i.e., which describe the configuration of M on input $\sigma_1 \ldots \sigma_n$ that runs up to step Cn^k (hence $1 \leq i, j \leq Cn^k$, $\gamma \in \Gamma$, and $q \in Q$). Then $\{x_{ij\gamma}, y_{ij}, z_{iq}\}$ are **deterministic** functions of $\sigma_1 \ldots \sigma_n$, and in particular:

(1) $x_{1j\gamma} = T$ for $j \leq n$ iff $\gamma = \sigma_j$;
(2) $x_{1j\gamma} = T$ for $n + 1 \leq j \leq Cn^k$ iff γ is the blank symbol;
(3) $y_{1j} = T$ iff $j = 1$, and $z_{1q} = T$ iff $q = q_0$;
(4) for each fixed i with $2 \leq i \leq Cn^k$, the variables $\{x_{i,j,\gamma}, y_{i,j}, z_{i,q}\}$ are deterministic functions of $\{x_{i-1,j,\gamma}, y_{i-1,j}, z_{i-1,q}\}$; and
(5) $\sigma_1 \ldots \sigma_n \in L$ iff $z_{Cn^k, q_{\mathrm{acc}}} = T$.

This allows us to build a circuit of size $C' n^{2k}$ that compute all $\{x_{ij\gamma}, y_{ij}, z_{iq}\}$, and therefore that computes $z_{Cn^k, q_{\mathrm{acc}}}$, which is T iff $\sigma_1 \ldots, \sigma_n \in L$.

[To complete the proof we need to fill in the details. See the proof of Theorem 9.30 of [Sip], pages 383–386.] $\square$

We remark that Theorem 9.30 of [Sip] is stated in more general terms, namely if $L \in \mathrm{TIME}(t(n))$ for a function t with $t(n) \geq n$ for all $n \in \mathbb{N}$, then L has circuits of size at most $C(t(n))^2$; also [Sip] uses the slightly different Boolean variables to prove the Cook-Levin theorem.

Corollary 3.4. *If* $\mathrm{P} = \mathrm{NP}$*, then for any* $L \in \mathrm{NP}$*, there are* $C, k \in \mathbb{N}$ *such that for all* $n \in \mathbb{N}$*,* $\mathrm{BoolFunct}_{L,n}$ *is expressed by a circuit of size at most* Cn^k*.*

Hence if you can prove that 3COLOR (described over $\{T, F\}$)—or any other NP-complete problem—does not have polynomial size circuits, then you have proved $\mathrm{P} \neq \mathrm{NP}$.

Next we state P versus NP as a problem in circuit complexity.

Notice that the circuits produced in Theorem 3.3 have a certain "uniform structure," given that they mostly consist of expressing, for each fixed i, the variables $\{x_{i,j,\gamma}, y_{i,j}, z_{i,q}\}_{j,\gamma,q}$ in terms of $\{x_{i-1,j,\gamma}, y_{i-1,j}, z_{i-1,q}\}_{j,\gamma,q}$.

Definition 3.5. Let $C_0, C_1, \ldots$ be a sequence of circuits, such that each $n \in \mathbb{N}$, C_n has n Boolean inputs (and therefore computes some function of these n inputs). We say that the family $\{C_n\}_{n\geq 0}$ is *P-uniform* if there is a Turing machine, M, that on input $\langle n\rangle$ produces a description of C_n and runs in time polynomial in n. (Therefore the size of C_n is at most some polynomial in n.)

It is not hard to prove the following variant of Corollary 3.4.

Theorem 3.6. *Let $L \subset \Sigma^*$ be NP-complete, where $\Sigma = \{T, F\}$ (or $\{0, 1\}$). Then $P = NP$ iff there is a P-uniform family of circuits $\{C_n\}_{n\geq 0}$ that computes* $\mathrm{BoolFunct}_{L,n}$.

Proof. If $\mathrm{P} = \mathrm{NP}$ then $L \in \mathrm{P}$, and we check that the circuits built by Theorem 3.3 are P-uniform. Conversely, if there is a P-uniform family $\{C_n\}_{n\geq 0}$ that computes $\mathrm{BoolFunct}_{L,n}$, then for any $w \in \Sigma^*$, we see whether or not $w \in L$ by building C_n with $n = |w|$, and then evaluate C_n on input $w = \sigma_1 \ldots \sigma_n$; this gives a polynomial time algorithm to test whether or not $w \in L$. □

The theorem above may seem unsatisfying, since the notion of P-uniform is stated in terms of Turing machines and polynomial time algorithms—it is difficult to state a more concrete description of what "P-uniform" means. Nonetheless, the above theorem is the way that many researchers tend to think of P versus NP, and, in particular, what it might take to prove that $\mathrm{P} \neq \mathrm{NP}$ (if this is really true).

3.5. **The Minimum Circuit Size Challenge.**

The minimum circuit size challenge is to determine the minimum circuit size for a given function $f\colon \{T, F\}^n \to \{T, F\}$. At present, the best lower bound for the minimum circuit size for an explicit function of n variables is a bound of order n.

SAY MORE ABOUT THE HISTORY AND/OR LITERATURE HERE.

4. Minimum Formula Size: Basic Results

In this section we make some easy observations about minimum formula size: namely, any Boolean function $\{T, F\}^n \to \{T, F\}$ can be expressed as a formula of size $n2^{n-1}$, and most such functions (i.e., more than half of the 2^{2^n} such functions) require a formula size at least $2^n/(4 + \log_2 n)$.

4.1. **Boolean Formula Size: Some Easy Remarks.**

Let us state some easy results regarding Boolean functions:

Proposition 4.1. *Let $n \in \mathbb{N}$.*

(1) There are 2^{2^n} (meaning, as usual, $2^{(2^n)}$) Boolean functions $f\colon \{T, F\}^n \to \{T, F\}$.
(2) Say that $f\colon \{T, F\}^n \to \{T, F\}$ is true on exactly m of its values. Then f can be expressed as a formula of size mn.

(3) *Say that $f\colon \{T,F\}^n \to \{T,F\}$ is false on exactly m of its values, Then f can be expressed as a formula of size mn.*
(4) *In particualr, any Boolean formula can be expressed with a formula of size at most $n2^{n-1}$.*

Let us indicate the proof:

(1) a Boolean function $f = f(x_1,\ldots,x_n)$ has 2^n values, one on each $(x_1,\ldots,x_n)\in\{T,F\}^n$, and there are two possible values (i.e., T or F).
(2) This is best illustrated by an example: say that $f\colon \{T,F\}^3 \to \{T,F\}$ is true on (T,T,T) and (F,F,T), and otherwise false; then we may write f as

$$f(x_1,x_2,x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \neg \wedge x_3); \tag{12}$$

the general result involves m clauses, each clause the AND of n literals representing a value where $f = T$.
(3) This follows by applying the previous result to the negation of f; for example, say that $g\colon \{T,F\}^3 \to \{T,F\}$ is false on (T,T,T) and (F,F,T), and otherwise true; then $g = \neg f$ where f is as in (12). Hence

$$g = \neg f = \neg\bigl((x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \neg \wedge x_3)\bigr),$$

and using (5) and (6) this becomes

$$g = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3).$$

(4) Any Boolean function is either true on at least $1/2$ of its 2^n values, or false on at least that many. Now apply (2) or (3).

Example 4.2. We have expressed Parity_2 as a formula of size 4; it is not hard to see that there is no smaller size formula for Parity_2. For $n = 2$, we have $n2^{n-1} = 4$; hence no other Boolean function of two variables requires a larger formula size than Parity_2.

4.2. Minimum Formula Size.

In the previous subsection we saw that every Boolean function $\{T,F\}^n \to \{T,F\}$ can be represented by a formula of size $2^n(n/2)$. The point of this subsection is to prove the following result.

Proposition 4.3. *For any $n \in \mathbb{N}$, more than half of the Boolean functions $\{T,F\}^n \to \{T,F\}$ cannot be represented by a formula of size less than $2^n/(4+\log_2 n)$.*

The proof simply counts the number of formulas of a given size.

Proof. The number of Boolean formulas of size s with operators $\wedge,\vee$ on the $2n$ literals $x_1,\neg x_1,x_2,\neg x_2,\ldots,x_n,\neg x_n$ is exactly equal to

$$\mathrm{Formulas}(n,s) \overset{\mathrm{def}}{=} C_s (2n)^s 2^{s-1},$$

where: C_s is the number of rooted, binary trees with s leaves; $(2n)^s$ reflects the s choices of $2n$ possible literals at the leaves; and the 2^{s-1} represents the 2 possible choices of $\wedge,\vee$ at each of the $s-1$ interior vertices. C_s is called a *Catalan number*,[3]

[3] C_s can be described in many equivalent ways, such as the number strings of length $2s$ of matching left- and right-parentheses.

and it is well-known that $C_s = \binom{2s}{s} \,/\, (s+1)$. Hence the majority of the 2^{2^n} Boolean functions on n variables cannot be expressed as a formula of size s provided that[4]

$$\text{Formulas}(n,1) + \ldots + \text{Formulas}(n,s) < (1/2)2^{2^n}. \tag{13}$$

Since C_s, $(2n)^s$, and 2^{s-1} are increasing functions of s (with n fixed), we have that Formulas(n,s) is increasing as a function of s; hence for $s \ge 2$,

$$\text{Formulas}(n,1) + \ldots + \text{Formulas}(n,s) \le s\ \text{Formulas}(n,s) = sC_s(2n)^s 2^{s-1}$$
$$= s\frac{1}{s+1}\binom{2s}{s}(2n)^s 2^{s-1} \le \binom{2s}{s}(2n)^s 2^{s-1}.$$

Note that $\binom{2s}{s} \le 2^{2s}$, since for any $k \in \mathbb{N}$ we have $\binom{k}{0} + \binom{k}{1} + \cdots + \binom{k}{k} = 2^k$. Hence

$$\text{Formulas}(n,1) + \ldots + \text{Formulas}(n,s) < 2^{2s}(2n)^s 2^{s-1},$$

and hence (13) holds whenever

$$2^{2s}(2n)^s 2^s \le 2^{2^n}.$$

Taking logarithms base 2, we can take any s with

$$2s + s(1 + \log_2 n) + s \le 2^n,$$

i.e., any s with $s \le 2^n/(4 + \log_2 n)$. □

5. The "Restriction" or "Random Restriction" Method of Subbotovskaya

The point of this section is to prove the following theorem.

Theorem 5.1 (Subbotovskaya, 1961)**.** *The function*

$$f(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$$

requires formula size at least $c\,n^{3/2}$*, for some constant* $c > 0$.

Our proof gives $c = 1/\sqrt{8}$; after the proof we give a more detailed calculation that shows that we can take $c = 3/2$ for $n \ge 3$ (and $c = \sqrt{2}$ for $n = 2$ and $c = 1$ for $n = 1$). [Hence the constant c is quite reasonable.]

It involves the idea of a "restriction" of formulas and functions.

5.1. Restriction.

If we take a Boolean function $f(x_1, \ldots, x_n)$ and fix some subset of $x_1, \ldots, x_n$ to Boolean values, what remains is a Boolean function of the remaining variables; we call this remaining function a *restriction of* f. Similarly, given any Boolean formula with variables $x_1, \ldots, x_n$, by restricting the values of some of $x_1, \ldots, x_n$ we get a new formula in terms of the other variables.

Example 5.2. Let $f(x_1, x_2, x_3)$ be given by the formula

$$f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_3).$$

By setting $x_1 = T$ we get the restriction

$$g(x_2, x_3) = f(T, x_2, x_3) = (T \wedge x_2) \vee (\neg x_3) = x_2 \vee (\neg x_3).$$

By setting $x_1 = F$ we get the restriction

$$g(x_2, x_3) = f(T, x_2, x_3) = (F \wedge x_2) \vee (\neg x_3) = F \vee (\neg x_3) = \neg x_3.$$

[4] Note that the function that is identically true could be viewed as a formula of size 0 (since it involves no variables), or as a size 2 formula $x_1 \vee \neg x_1$; similarly for the identically false function. Hence, we should really insist that $s \ge 2$, or add 2 to the left-hand-side into (13).

By setting $x_3 = F$ we get the restriction

$$g(x_1, x_2) = f(x_1, x_2, F) = (x_1 \wedge x_2) \vee T = T.$$

Note that in the above example, the formula obtained by restriction can sometimes be further simplified, by repeatedly applying the simplification rules:

$$T \wedge x = F \vee x = x, \quad T \vee x = T, \quad F \wedge x = F$$

to the restricted formula. So the formula $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ simplies to F under the restriction $x_1 = F$, but only simplifies to $x_2 \wedge \ldots \wedge x_n$ under the restriction $x_1 = T$.

5.2. **Subbotovskaya's Fundamental Lemma.**

Lemma 5.3 (Subbotovskaya's Restriction Lemma)**.** *Let $n \geq 2$ and $f(x_1, \ldots, x_n)$ be any Boolean function whose minimum formula size is L. Then there exists a restriction of f by fixing a single variable, such that the resulting function of the remaining $n-1$ variables has size at most $L(1 - (3/2)/n)$.*

In fact, much more is true: if we pick one of the n variables $x_1, \ldots, x_n$ at random, and randomly set it to T, F (each with probability $1/2$), then the *expected* or *average* size of the remaining formula is at msot $L(1 - (3/2)/n)$. For this reason Subbotovskaya's method is often called the method of "random restrictions."

To prove Lemma 5.3, we need a few preliminary results.

Lemma 5.4. *Let $g(x_1, \ldots, x_n)$ be an arbitrary Boolean function. Then*

$$x_1 \wedge g(x_1, x_2, \ldots, x_n) = x_1 \wedge g(T, x_2, \ldots, x_n)$$

(this equation is an equality of Boolean functions). Similarly

$$\begin{aligned} \neg x_1 \wedge g(x_1, x_2, \ldots, x_n) &= x_1 \wedge g(F, x_2, \ldots, x_n), \\ x_1 \vee g(x_1, x_2, \ldots, x_n) &= x_1 \vee g(F, x_2, \ldots, x_n), \\ \neg x_1 \vee g(x_1, x_2, \ldots, x_n) &= x_1 \vee g(T, x_2, \ldots, x_n). \end{aligned}$$

Proof. For any values of $(x_1, \ldots, x_n) \in \{T, F\}^n$, we have $x_1 \wedge g(x_1, x_2, \ldots, x_n) = T$ iff $x_1 = T$ and $g(x_1, x_2, \ldots, x_n) = T$, which holds iff $x_1 = T$ and $g(T, x_2, \ldots, x_n) = T$. Similarly for the other formulas. $\square$

Lemma 5.5. *$F = F(x_1, \ldots, x_n)$ is a minimal size formula for a Boolean function, $f = f(x_1, \ldots, x_n)$, and a leaf x_i of F is connected to an interior node v_j of F, then the other input to v_j is a formula, g, that is independent of x_i.*

We illustrate this lemma in Figure 5.

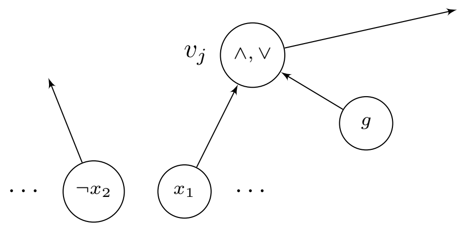


FIGURE 5. g is independent of x_1 in a minimum size formula.

Proof. If not, Lemma 5.4, with x_1 replaced with x_i everywhere, can be used to produce a smaller formula. □

Proof of Lemma 5.3. For some $i \in [n]$, at least $t = L/n$ of the leaves equal $x_i, \neg x_i$; choose any such i. Each leaf x_i is connected to an interior node v_j, whose other input, g, is independent of x_i.

Say that v_j computes the function $x_i \wedge g$, then setting $x_i = T$ and $x_i = F$, respectively, we are respectively left with the functions g and F; hence the vertex x_i always eliminated from the formula tree, and for one of $x_i = T, F$, all the children of g (i.e., leaves in the subtree of vertices that eventually lead to g) are eliminated. Hence on average we eliminate $3/2$ inputs for each such interior node v_j.

The same argument with v_j computing $\neg x_i \wedge g$, $x_i \vee g$, and $\neg x_i \vee g$ shows that the average number of leaves eliminated when setting x_1 to T and F is $3/2$ per interior node that has x_1 as an input. Hence by setting x_i to T or F we get a tree with at most $L(1-(3/2)/n)$ leaves remaining. □

5.3. **Proof of Subbotovskaya's Theorem.**

Proof of Theorem 5.1. By Subbotovskaya's Lemma, there is a one-variable restriction of the formula for $\mathrm{Parity}_n(x_1, \ldots, x_n)$ of size at most $L(1-(3/2)/n)$. Letting L' be the smallest size formula for this restriction, and taking another restriction, there is a formula for some two-variable restriction of Parity_n of size at most

$$L\left(1-\frac{3/2}{n}\right)\left(1-\frac{3/2}{n-1}\right).$$

Repeating this argument shows that for any $m \le n$ there is an $(n-m)$-variable restriction of Parity_n of size at most

$$L\left(1-\frac{3/2}{n}\right)\left(1-\frac{3/2}{n-1}\right)\ldots\left(1-\frac{3/2}{m+1}\right) \ge L_m \tag{14}$$

where L_m is the minimum formula size for Parity_m. Now take $m=1$; the resulting Parity_1 function depends on its single variable, and hence of size at least one. Hence

$$L\left(1-\frac{3/2}{n}\right)\left(1-\frac{3/2}{n-1}\right)\ldots\left(1-\frac{3/2}{2}\right) \ge 1,$$

so

$$L \ge \left(1-\frac{3/2}{n}\right)^{-1}\ldots\left(1-\frac{3/2}{2}\right)^{-1}.$$

Taking logarithms base e and using the fact that $\log(1-\epsilon) \le -\epsilon$ (which can be seen from a Taylor expansion), we have

$$\log_e L \ge (3/2)\left(\frac{1}{2}+\frac{1}{3}+\cdots\frac{1}{n}\right),$$

$$\ge (3/2)\int_2^{n+1} \log_e(x)\,dx \ge (3/2)\bigl(\log_e(n+1)-\log_e 2\bigr)$$

(where we have used the integral bound on the sum $1/x$ from $x=2$ to $x=n$). Hence

$$L \ge (n+1)^{3/2}/2^{3/2} \ge n^{3/2}/\sqrt{8}.$$

Hence $L \ge c\,n^{3/2}$ with $c = 1/\sqrt{8}$. □

We remark that one can easily improve the constant c above: indeed, the proof above implies that if L_n is the size of smallest formula for Parity_n, then

$$L_n \geq h(n) \overset{\mathrm{def}}{=} \prod_{i=2}^{n} \left(1 - \frac{3/2}{i}\right)^{-1}.$$

Defining c_n via $h(n) = c_n n^{3/2}$, a short calculation shows that c_n is monotone increasing over integers $n \geq 1$.[5] Seeing as $c_1 = 1$, $c_2 = \sqrt{2}$, and $c_3 = 1.539\ldots$, we can take $c = 3/2$ for $n \geq 3$ (a Python calculation suggests $c_{1000000} = 1.77245318\ldots$).

Appendix A. Some Progress in Formula Size After Subbotovskaya

Research into formula size has seen a slow but steady improvement since 1961; here we focus our remarks on Subbotovskaya's method.

A.1. **Shrinkage.**

Subbotovskaya's proof shows that all one-variable restrictions of a formula on n variables causes it size to drop by a factor of $1 - (3/2)/n$. Imagine that one can improve this to a factor of $1 - \omega/n$ for $\omega > 3/2$. Then after restricting $n - m$ of its variables, the average size of the formula that remains is a formula on m variables that shrinks in size by a factor of

$$\bigl(1 - \omega/n\bigr)\bigl(1 - \omega/(n-1)\bigr)\ldots\bigl(1 - \omega/(m+1)\bigr),$$

which is bounded above and below by a constant times $(n/m)^{-\omega}$. So if the average restriction of m variables is a formula of size at least L_m, then the original formula must have been of size at least $(n/m)^{\omega} L_m$.

Definition A.1. The *shrinkage exponent* is the supremum, ω, of all real ω' such that for any $m < n$, the size of any Boolean formula on n variables of size L is reduced to size at most $cL(m/n)^{\omega'}$ for some constant c (independent of n, m, L).

We remark that Subbotovskaya's result implies that the shrinkage exponent is at least $3/2$. Since Parity_n has formulas of size $O(n^2)$, this implies that the shrinkage exponent is at most $\omega \leq 2$, since $L_1 = 1$ and $L_n \leq 4n^2$, where L_i is the minimum formula size for Parity_i.

It wasn't until 1993 that Impagliazzo and Nisan [IN93] improved the shrinkage exponent to $\omega \geq 1.55$; slightly later Paterson and Zwick [PZ93] improved this to $\omega \geq 1.63$; in 1998 Håstad improved this to $\omega = 2$, settling the value of the shrinkage exponent.

[5] To see this, note that

$$\bigl(c_n/c_{n-1}\bigr)^2 = \frac{(n-1)^3}{n^3}\left(1 - \frac{3/2}{n}\right)^{-2} = f(1/n),$$

where $f(x) = (1-x)^3/(1-(3/2)x)^2$. Hence c_n is increasing in n provided that $f(x) \geq 1$ for $0 \leq x \leq 1/2$; to show this, it suffices to show that $g(x) = (1-(3/2)x)^2 - (1-x)^3$ satisfies $g(x) \leq 0$ in this range. Since $g(0) = 0$, and

$$g'(x) = 2(1-(3/2)x)(-3/2) + 3(1-x)^2 = 3\bigl(-1 + (3/2)x + (1-x)^2\bigr) = 3(x^2 - x/2) = 3x(x - 1/2);$$

hence $g'(x) < 0$ for $0 < x < 1/2$; and hence $g(x) < 0$ for $0 \leq x \leq 1/2$, and hence $f(x) \geq 1$ in this range of x; hence c_n is monotone increasing.

A.2. **Andre'ev's Function.** In 1987, Andre'ev [And87] constructed a (polynomial time computable) function on n variables (for a sequence of n tending to infinity) whose restriction from n variables to $m = n^\epsilon$ variables, for any $\epsilon > 0$, reduces to a formula of size roughly order $n/\log(n)$. As a consequence, the minimum formula size of Andre'ev's function is at least $n^{1+\omega}/\log(n)$. Combining this with Håstad's result, this proves that Andre'ev's function requires a size of $n^{3-\epsilon}$ for any $\epsilon > 0$.

Andre'ev [And87] cites [Sub61, Neč66] as inspiration. Specifically the rough idea behind *Nechiporuk's theorem* [Neč66] is that if a Boolean function encodes a "large number" of functions on a smaller set of variables, then this can give interesting formula size lower bounds.

Here is Andre'ev's construction.

Theorem A.2 (Andre'ev)**.** *For any $\epsilon > 0$ there is a $c > 0$, a sequence of integers n tending to infinity, and a function $A(x_1, \ldots, x_n)$ (computable in polynomial time) such that most (more than half of the) random restrictions of $n - n^\epsilon$ variables of $A(x_1, \ldots, x_n)$ (i.e., leaving n^ϵ of its variables unrestricted) require a formula of size $\geq cn/\log_2 n$.*

Proof. Let us sketch the idea. For each $N \in \mathbb{Z}$ define n via $n/2 = 2^N$. Hence each function $\{T, F\}^N \to \{T, F\}$ can be specified using Boolean variables $x_1, \ldots, x_{n/2}$; specifically if $z_1, \ldots, z_N$ are N Boolean variables, and $z_1 \ldots z_N$ is the binary representation of a number between 0 and $2^N - 1 = (n/2) - 1$, then $(z_1, \ldots, z_N) \mapsto x_{(z_1 \ldots z_N)+1}$ is a Boolean function; as $x_1, \ldots, x_{n/2} = x_{2^N}$ vary over all values of T, F, $x_{(z_1 \ldots z_N)+1}$ varies over all 2^{2^N} Boolean functions on $z_1, \ldots, z_N$.

The next step in Andre'ev's construction is to divide $x_{n/2+1}, \ldots, x_n$ into N groups of nearly equal size (i.e., either $\lfloor n/(2N) \rfloor$ or $\lceil n/(2N) \rceil$, and note that either is roughly $n/2\log_2(n)$); and let $z_1, \ldots, z_N$ be the partity of each of these N groups of variables. So one can take Andre'ev's function to be

$$A(x_1, \ldots, x_n) = x_{(z_1 \ldots z_N)+1}.$$

Now consider a random restriction of $A(x_1, \ldots, x_n)$ of $n - n^\epsilon$ variables, leaving a function of n^ϵ of the variables $x_1, \ldots, x_n$. Now let us make some additional restrictions.

First, take a random restriction of any other variables in $x_1, \ldots, x_{n/2}$ that are unrestricted. This means that $x_1, \ldots, x_{n/2}$ describe a random function $\{T, F\}^N \to \{T, F\}$, which is therefore requires an expected formula size of at least $2^N/(8 + 2\log_2 N)$ (since the majority of the 2^{2^N} functions require size at least $2^N/(4 + \log_2 N)$.

Next, a short calculation shows that in all the $z_1, \ldots, z_N$, it is likely that each z_i contains at least one variable unrestricted. Given this, let us further restrict the $x_{n/2+1}, \ldots, x_n$ so that exactly one variable in each of groups defining $z_1, \ldots, z_n$ is unrestricted. The $x_{(z_1 \ldots z_N)+1}$ is a random function on N variables, and hence requires a formula of size at least $2^N/(8 + 2\log(N))$ (with high probability). □

A.3. **Other Historical Remarks.** The subject of formula size is a large field of study.

Regarding minimum formula size, in 1966, Nechiporuk [Neč66] produced a formula requiring $c(n/\log n)^2$ size formulas; the general lower bounding method is called *Nechiporuk's theorem.* in 1971, Khrapchenko [Hra71] proved that Parity_n requires $\geq cn^2$ size formulas by a different method called *Khrapchenko's method.*

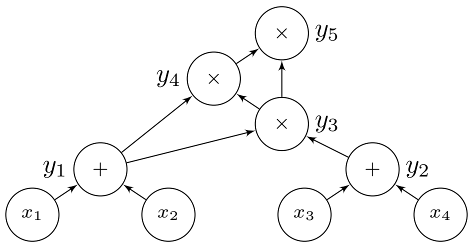
For more historical remarks up to roughly 1987, see also Wegener's "blue book" [Weg87].

A.4. **Recent Research, Additional Remarks.** Talk about communication complexity and the attempt to break the n^3 barrier.

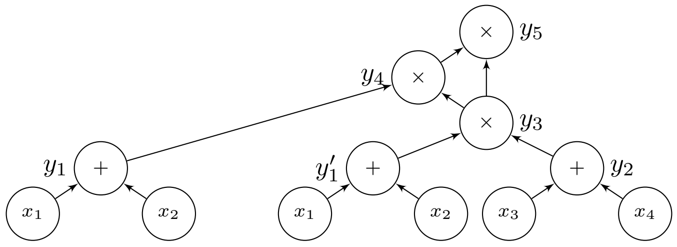
Appendix B. From Circuits to Formulas

In this section we explain how to take a circuit and produce a formula of the same depth that computes the same Boolean function.

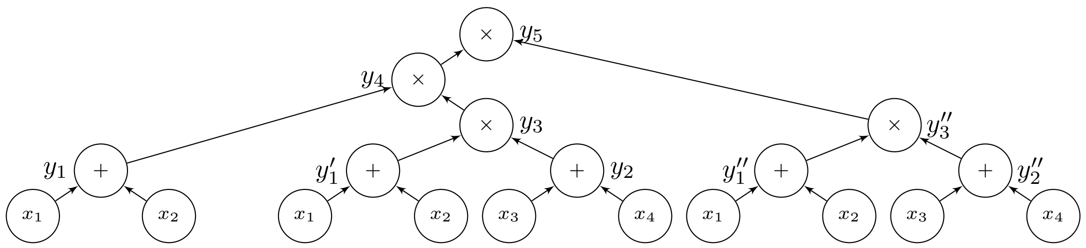
Note that circuit in Figure 4,



can be converted to an equivalent formula, by replicating interior vertices that are used more than once, i.e., that have more than one outgoing arrow. The general procedure can be understood by studying the example above: first we introduce $y_1' = y_1$ to take care to the two outgoing arrows at y_1:



so now y_1' replacates y_1 (and the entire tree beneath it), and the vertices labelled y_1, y_1' each have a singe outgoing edge; then we introduce $y_3'' = y_3$, and replicate then entire subtree at and below y_3, which yields:



Since each vertex has outdegree 1 except for the root, this circuit represents a tree and the formula

$$\Big((x_1+x_2)\big((x_1+x_2)(x_3+x_4)\big)\Big)\Big(\big((x_1+x_2)(x_3+x_4)\big)\Big),$$

which is of size 10, which has 9 operations in total, which is essentially (3) with some multiplications exchanged.

Remark B.1. The number of operations $+, -, \times$ in a formula is exactly one less than the number of leaves or—in the last case—the number of x_1, x_2, x_3, x_4 appearing in the formula.

Remark B.2. Note that the circuit and resulting formula have the same *depth*, i.e., the same maximum distance between the root and all leaves; in the above example the depth is 4.

References

[And87] A. E. Andreev, *On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes*, Vestnik Moskov. Univ. Ser. I Mat. Mekh. (1987), no. 1, 70–73, 103. MR 883632

[Hra71] V. M. Hrapčenko, *The complexity of the realization of a linear function in a class of Π-schemes*, Mat. Zametki **9** (1971), 35–40. MR 290872

[IN93] Russell Impagliazzo and Noam Nisan, *The effect of random restrictions on formula size*, Random Struct. Algorithms **4** (1993), no. 2, 121–134.

[Neč66] È. I. Nečiporuk, *On a Boolean function*, Dokl. Akad. Nauk SSSR **169** (1966), 765–766. MR 218148

[PZ93] Mike Paterson and Uri Zwick, *Shrinkage of de morgan formulae under restriction*, Random Struct. Algorithms **4** (1993), no. 2, 135–150.

[Sub61] B. A. Subbotovskaya, *Realization of linear functions by formulas using $\vee$, &, $-$*, Dokl. Akad. Nauk SSSR **136** (1961), 553–555, Translation in *Soviet Math. Dokl.*, **2** (1961), 110–112,.

[Weg87] Ingo Wegener, *The complexity of Boolean functions*, Wiley-Teubner Series in Computer Science, John Wiley & Sons Ltd., Chichester, 1987. MR 905473 (89b:03066)

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`