DRAFT: THE COOK-LEVIN THEOREM AND HOW TO SOLVE (AND NOT SOLVE) P VERSUS NP, AND WHAT COMES AFTER CPSC 421/501

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 536F: use this material at your own risk...

The point of this article is to identify some problems that can be viewed as the "long-term goals" in algorithms and *complexity theory* (which typically focuses on "lower bounds" on the resources required by algorithms that solve certain problems). In doing so, we wish to explain the connection to topics studied in CPCS 421/501, Fall 2021. In particular, we introduct some aspects of the Mulmuley-Sohoni approach to Permanent Versus Determinant and our simplification an variants of their approach that will motivate part of the course material in CPSC 536F. This includes the classical problems of circuit/formula size/depth

Date: Wednesday 8$^{\text{th}}$ December, 2021, at 11:28(get rid of time in final version).

Research supported in part by an NSERC grant.

required to compute certain functions, some of which has complete analogs in the Boolean and algebraic settings.

At present it is unclear how much progress has been made on these "long-term goals," but there are certainly a lot of interesting ideas out there that are worth understanding and pursuing.

As always, please let me know if there corrections needed. And, needless to say, the notes here are based on my recollections and particular experiences, etc.

1. Overview

The these notes have a few main goals:

(1) to conceptually simplify the proof of the Cook-Levin Theorem in [Sip], Chapter 7, and to make some remarks about the approaches to solving P versus NP in Chapter 9 there; and
(2) to indicate what topics are typically studied in a second course in complexity theory; at any point in time this depends on current approaches to solving questions like P versus NP (and many presumably much easier questions).

The proof of the Cook-Levin Theorem was one of the surprises of computation theory at the time; it tends to be taken for granted these days as a starting motivation for studying the problem P versus NP. Using the notation for Turing machine configurations of [Sip], Chapter 3, the Cook-Levin Theorem shows the remarkable power of grid of squares—each connected to only a few of its neighbours—to create a coherent global computation[1].

Unfortunately, the typical proof of the Cook-Levin theorem tends to be buried underneath a lot of Boolean algebra, which obscures the very elegant and fundamental idea behind the proof. The main point of these notes is to conceptually simplify the proof.

[There is a much longer story here: many classical differential equations[2] and differential operators[3] have a similar way of having local structure organize a global structure. Also there is a lot of work in theoretical physics and probability (some called percolation theory, Brownian motion, stochastic PDE's, etc.)—that similarly studies the extent to which local phenomenon can have interesting long-range effects, and connect to other fields.]

It is also interesting—historically—to see how the "iron curtain" of the "cold war" that developed in the mid 1940's to roughly the 1990's (?) created a system whereby "the East" and "the West" were not sharing information, which lead to independent research developments that mirrored each other, although sometimes

[1] A 2-dimensional in space (and 1-dimensional in time) version of this idea was popularized as a "game of life," attributed to John Horton Conway (perhaps others?), around the same time as the works of Cook and Levin. However, the works of Cook and Levin concern a single-tape Turing machine, which is 1-dimensional in space, and hence more difficult to work with than a two or higher dimensional version; the version of Conway (et al.?) can be made more suggestive of biological creatures; likely a three (or larger) space dimension version (coupled with some form of virtual reality tech) could be even more compelling.

[2] e.g., ODE's, the classical heat equation, etc.; things are a bit different if morally fractional derivatives are involved, such as the classical wave equation $u_{tt} = \Delta u$ in odd space dimensions.

[3] local operators, not, say, pseudo differential operators, invented to study oscillatory integrals, create a reasonable foundation for quantum mechanics, etc.

separated in time by years if not decades. We make some brief remarks that hopefully are fair to everyone involved and will also amuse the reader (one can always hope. . .).

2. The Setup of the Cook-Levin Theorem as a Theorem on Local Behaviour

Consider the style of configuration representation in Figure 3.4 of [Sip], which represents this Turing machine configuration as $1011q_7 0111$, which represents the configuation that is (1) in state q_7, (2) has the tape head over cell number 5, and (3) has a single tape with contents 10110111 followed by infinitely many "blank" cells.

More genearlly one is representing a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\rm acc}, q_{\rm rej})$ configuration as a string over an alphabet $\Sigma_{\rm config} = Q \cup \Gamma$, assuming these sets are disjoint[4].

The remarkable fact about viewing a configuration in this way is that we easily check that the definition of how a Turing machine works implies that for any $t = 0, 1, \ldots,$ if at step t and $t + 1$ we are respectively in configurations

$$\sigma_1\sigma_2 \ldots, \quad \sigma'_1\sigma'_2 \ldots,$$

then σ'_k can be expressed solely as a function of $\sigma_{k-1}, \sigma_k, \sigma_{k+1}$. Hence for any fixed Turing machine, M, its behaviour can be described by (1) its input, which determines its configuration in step 0, and (2) a transition rule

$$\sigma'_k = {\rm trans}_M(\sigma_{k-1}, \sigma_k, \sigma_{k+1}), \tag{1}$$

applied to the k-th cell, σ'_k, at time $t+1$, to obtain the result of a single step of the machine from the time t from cells $k-1, k, k+1$, with a fixed function

$${\rm trans}_M : \Sigma^3_{\rm config} \to \Sigma_{\rm config}.$$

If we are interested in the configuration at the end of and step T, it suffices to observe cells 1 through $T+1$ in steps $t = 0, 1, \ldots, T$ (since cells $T+2, T+3, \ldots$ are necessarily blank; see Figure 7.38, which delimits adds some extra # characters for clarity with $T = n^k$, and n the length of the input).

3. Verifiers, NP, and Statement of the Cook-Levin Theorem

In class we reviewed the language SAT, and the classes P (of languages that can be recognized by a Turing machine in polynomial time), and NP (of languages verifiable in polynomial time).

NP is traditionally defined as the class of languages, L over some alphabet, that can be decided by a non-deterministic Turing machines runs in polynomial time; a more modern—and perhaps more elegant—way to define this in terms of *verifiers*. Let us recall roughly this means in the context of NP.

Let $p\colon \mathbb{Z}_{\ge 0} \to \mathbb{N}$ be any function (of course, we think of p as a polynomial when thinking of NP, the class of languages verifiable in polynomial time). We say that a languages L over an alphabet Σ can be *verified* within time $p(n)$ if there exists a

[4] This runs into trouble in, say, standardized Turing machines, where by convention Q and Γ are each taken to be the first so many integers, and so Q and Γ must intersect. When Q and Γ do intersect, one takes $\Sigma_{\rm config} = Q \amalg \Gamma$, the *disjoint union*, a limit that is unique up to unique isomorphism, but we can ignore this point in this article.

Turing machine (called a *verifer* of L) M such that for all $w \in \Sigma^*$, setting $n = |w|$ (the length of w), we have

(1) if $w \in L$, then there exists a string $g = (g_1, \dots, g_{p(n)-n})$ such that on input wg, M accepts wg in time at most $p(n)$, and
(2) if $w \notin L$, then there exists no string $g = (g_1, \dots, g_{p(n)-n})$ such that on input wg, M accepts wg in time at most $p(n)$.

The reason that we limit the "guess" g to $p(n) - n$ symbols is that if we run the algorithm for $p(n)$ steps in total, the computation will only depend on the first $p(n)$ symbols written on the tape; note that one can let $p(n) < n$, in which case we can ignore the guess, and the Turing machine doesn't even need to look at the input symbols.

We then used NP to denote the class[5] of languages that can be verified in polynomial time, i.e., time $O(n^k)$ for some $k \in \mathbb{N}$. This is the approach taken in [Sip], Definition 7.19; then [Sip] defines non-deterministc Turing machines, and proves in Theorem 7.20 that a language is in NP iff it can be decided by some non-deterministic Turing machine in polynomial time. See also Exercise **??** for a standard illustration of algorithms that can take advantage repeated calls to a SAT oracle.

Theorem 3.1 (Cook-Levin)**.** *If* SAT $\in P$*, then P=NP. More precisely, given an oracle for SAT over the alphabet*

$$\Sigma_{\mathrm{SAT}} = \{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, x, 0, 1 \dots, 9, (,)\}$$

and any language $L \in NP$, there is an algorithm, $M_{\text{oracle SAT}}$ or M^{SAT}, i.e., an oracle Turing machine, that on any input of length n can verify *whether or not $w \in L$ via (1) some preprocessing that requires at most time polynomial in n, and then (2) makes a single oracle call to SAT. Similarly with 3SAT replacing SAT.*

Even more precisely, one can state that the preprocessing required takes *space* $O(\log(n))$, which is a far more restrictive condition than taking time polynomial in n (this is implicit in Chapter 8 of [Sip]). In class on November 30, 2021, we stated only the first form of Theorem 3.1, and outlined the proof based on the second form.

Classically the more precise version of Theorem 3.1 is used to define what is meant by *NP-completeness*, and then one states the Cook-Levin theorem using NP-completeness. However, this is essentially an accident of classes P and NP: in practice we are unlikely to care how many times the oracle for SAT is called and when, as long as the resulting algorithm is practical (and even if it isn't, as a possible intermediate step to construct something practical).

The motivation for showing that it suffices to have an oracle for 3SAT rather than for SAT is that it is much simpler to reduce a 3SAT question to many standard NP-complete languages (e.g., 3COLOUR, SUBSET-SUM, etc.). Hence one can easily then show that 3COLOUR, SUBSET-SUM, etc., are also examples of a "hardest problem" in NP. See the numerous examples in Chapter 7 of [Sip].

4. The Proof of the Cook-Levin Theorem

In this section we outline the proof of the Cook-Levin Theorem.

[5] One should refer to this as a *class*, not a *set*, in common set theory conventions in 2011 unless one, say, fixes the alphabet Σ or assumes that Σ is standardized.

Let $p\colon \mathbb{Z}_{\geq 0} \to \mathbb{N}$ be any function (for the proof we want p to be bounded by Cn^k for some C, k, but for now we can leave the discussion general. Let L be verifiable in time $p(n)$, and let $M = (Q, \Sigma, \Gamma, \delta, q_{\mathrm{init}}, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ be a verifier for L that runs in time $p(n)$ (and we assume $p(n) \geq n$ for all n). By definition, for any $w \in \Sigma^*$, we have $w \in L$ iff

$$\exists g \in \Gamma^{p(n)-n} \quad \text{such that} \quad wg \text{ is accepted by } M.$$

So consider a Turing machine that has wg on its input tape initially, i.e., at "step 0," and runs for $p(n)$ steps. Even if the tape head moves right at each step, it will only reach cell $p(n)$ after $p(n) - 1$ steps, and hence it is only cells 1 through $p(n)$ that can affect the computation.

Using the configuaration representation in Section 2, the step 0 configuration is

$$q_{\mathrm{init}}\ w_1\ \ldots\ w_n\ g_1\ \ldots g_{p(n)-n}.$$

So for $i = 1, \ldots, p(n) + 1$, and $t = 0, \ldots, p(n)$, let $\mathrm{symb}_{i,t}$ denote the element of $Q \cup \Gamma$ corresponding to the i-th letter (counting left-to-right) at time t. This "square grid of cells" was depicted in class. [Note that Figure 7.38 of [Sip] is very similar, except that [Sip] is using a non-deterministic Turing machine, so the start configuation there has only w following by blanks, and the proof is a variant of the proof we give.] The machine M accepts wg iff

(1) the symbol $\mathrm{symb}_{1,0} = q_{\mathrm{init}}$ and for $i \geq 2$, $\mathrm{symb}_{i,0}$ is the $i - 1$ symbol of wg;
(2) for $t = 1, \ldots, p(n)$ and $i = 2, \ldots, p(n) - 1$ we have
$$\mathrm{symb}_{i,t} = \mathrm{trans}_M\bigl(\mathrm{symb}_{i-1,t-1}, \mathrm{symb}_{i-1,t}, \mathrm{symb}_{i+1,t-1},\bigr),$$
where trans_M is the transition function in (1); and
(3) for some i we have $\mathrm{symb}_{i,p(n)} = q_{\mathrm{acc}}$

(here, for simplicity, we keep the Turing machine in q_{acc} until step $p(n)$ if enters this state earlier, and similarly for q_{rej}). Hence we can write the condition $w \in L$ as iff there is a way of choosing values to the variables

$$g_1, \ldots, g_{p(n)-n} \in \Gamma, \quad \mathrm{symb}_{1,0}, \mathrm{symb}_{2,0}, \ldots, \mathrm{symb}_{p(n)+1,p(n)} \in Q \cup \Gamma$$

such that the conditions (1)–(3) hold. Since $g_i = \mathrm{symb}_{i-1,0}$, one can write this down as a set of logical conditions involving only the variables $\mathrm{symb}_{i,t}$ and the function trans_M.

Then one reduces this this to a Boolean formula by introducing the Boolean variables $x_{i,t,s}$ for $s \in Q \cup \Gamma$ where $x_{i,t,s}$ is true iff $\mathrm{symb}_{i,t} = s$. The task, which is a bit tedious but not very difficult, is to show that one can write down a Boolean formula of size polynomial in n whose satisfiability is equivalent to the conditions above on the variables $\mathrm{symb}_{i,t}$.

Let us indicate how one does this.

There are a few things to note: first, this gives $(p(n) + 1)(p(n) + 1)|Q \cup \Gamma| = O(p(n)^2)$ Boolean variables $x_{i,t,s}$; hence the total number of Boolean variables $x_{i,t,s}$ is a polynomial in n (that depends on M and $p(n)$, which are both fixed features of our verifying algorithm).

Next the the choice of a value of $\mathrm{symb}_{i,t}$ for a fixed i, t requires $x_{i,t,s}$ to be true for exactly one value of s, which we can write down as a formula to be satisfied that is of constant length (the constant depending on M) for each i, t.

One similarly writes down formulas for conditions (1)–(3) above to hold.

With a bit trickier Boolean algebra, one can write this formula in 3CNF form. It is pretty clear that when combining a bunch of conditions together, each of which must be satisfied individually, then this is the AND (or conjunction) of some smaller formulas. However, there are a few clever tricks of Boolean algebra involved in reducing each of the smaller formulas in 3CNF form, so that ulmitately one can combine all the conditions into one large 3CNF formula. One of them is that a condition like

$$x_1 \vee x_2 \vee \ldots \vee x_m$$

(with $m \geq 4$) can be written in an equivalent 3CNF satisfiability form by introducing new variables $y_1, \ldots, y_{m-3}$ and writing the equivalent condition

$$(x_1 \vee x_2 y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge \ldots (\neg y_{m-3} \vee x_{m-1} \vee x_m).$$

Hence one concludes that not only can we reduce the question of whether or not $w \in L$ to the question of determining whether or not a formula of size polynomial in $|w|$ is satisfiable, but we may also write this formula in 3CNF form.

5. UNDER CONSTRUCTION: Continuations of CPSC 421/501 At UBC

In the early- and mid-1990's, CPSC 501 was a type of sequel to CPSC 421. What has survived as a true second course in complexity theory is CPSC 536, usually offered once every two years. It covers what is a standard next course in modern complexity theory, a true continuation of the study of P versus NP and related (likely far easier questions). This includes the alternative problem of determining whether or not SAT (or any other NP-complete problem) can be computed by uniform circuits, and the more difficult but simpler problem of determining which Boolean functions $\mathbb{B}^n \to \mathbb{B}$ can be computed by circuits (not necessarily uniform) of a given size.

INSERT PLUG FOR MY CPSC 531F and CPSC 536F TOPICS COURSES HERE, BASED ON THE LAST SECTION.

6. UNDER CONSTRUCTION: History: A Few Recent Approaches to P Versus NP as of 2021

The material here will not be covered per se in CPSC 421/501, but will indicate what a second course in theoretical computer science maight cover (e.g., CPSC 536, other topics courses in CS theory).

Many people in computer science are familiar with the statement of the problem P versus NP and have given the matter some consideration. This question has given rise to a number of very important developments in algorithms and *complexity theory*, i.e., the study of algorithms with restricted resources (e.g., Turing mathcines with time and space restrictions, the simplified model of DFA's and NFA's, etc.).

There are few attitudes I have heard to the problem of P versus NP:

(1) one can study problems cannot be solved in polynomial time; this seems intrinsically interesting, since a lot of algorithms take polynomial time (e.g., many algorithms encountered in CPSC 340, algorithms based on *dynamic*

programming[6]). Of course, the idea of polynomial time is to avoid algorithms that become impractical with large input size; one can similarly study algorithms with related limitations, giving rise to what one calls *complexity theory.* One can, of course, explore algorithms that—while not provably in polynomial time—seem to work well in practice.

(2) One can seek to develop better algorithms that may solve problems like SAT well in practice and/or possibly provably in polynomial. I don't know if people actually believe that P=NP, or if stating this view is a sort of "contrarian" point of view, and/or a cry to develop more sophisticated algorithms in general, etc.

(3) P versus NP might be unresolvable in our current commonly used systems of set theory; I heard this possibility from the computer science theoretician William Gasarch in the early-mid 1980's, and I don't know how common a belief this is, or if this just a "contrarian" point of view.

However, simply because a problem has received intense study from the theoretical computer science and community and algorithms community (narrowly or broadly interpreted) for some 50+ years, doesn't really mean that one can say very much.

Here we will discuss some problems related to P versus NP that—albeit likely much easier—are still wide open as of 2021, and concern finding "lower bounds" in complexity theory, attempts to show that certain problems cannot be solved with certain constrained resources.

6.1. **The Mulmuley-Sohoni Approach.** In [MS01], Ketan Mulmuli and Milind Sohoni proposed an approach to P versus NP called Geometric Complexity Theory. Their approach gives an intruiging approach to study the well-known open problem of *Permanent Versus Determinant*. There is a lot of further speculation in this article; let us content ourselves with remarks on Permanent versus Determinant, which is an "algebraic analog of NC Versus NP," or, due to a collapse in algebraic complexity theory, an analog of "poly-log time versus NP."

Permanent Versus Determinant can be be understood as follows: let $R = \mathbb{Z}/2\mathbb{Z}$, or, more generally, let R be any fixed ring. For any $n \in \mathbb{N}$, let $p = p(n)$ be the smallest integer such that one may express the $n \times n$ permanent $\mathrm{Perm}(a_{ij})$ as a $p(n) \times p(n)$ determinant whose entries are, say, linear functions of the a_{ij},

$$\mathrm{Perm}(a_{ij}) = \mathrm{Det}\bigl(\ell_{IJ}(a_{ij})\bigr)$$

The motivation of this problem is ETC.

Their approach uses the fact that the determinant function $\mathrm{Det}(a_{ij})$ has a lot of symmetries. It follows that if $\mathrm{Perm}(a_{ij})$ can be expressed as a $p(n) \times p(n)$ determinant, and if we enlarge the matrix a_{ij} to be a $p(n) \times p(n)$ set of variables (using I, J to denote indices in $[p(n)] = \{1, \ldots, p(n)\}$, then

$$g(\epsilon; a_{ij}) \overset{\mathrm{def}}{=} \mathrm{Det}\bigl(\ell_{IJ}(a_{ij}) + \epsilon a_{IJ}\bigr)$$

satisfies:

[6] This approach is based on work at the RAND Corporation in the late 1940's and early 1950's of the founder of the modern theory of differential games, Rufus Isaacs, and of a pioneer in the theory of optimal control, Richard Bellman. Given the lack of transparency of their collaboration, it is difficult to assign any more precise attribution here. In the context of algorithms, this often leads to a *cubic time* algorithm, i.e., in the natural measure of the problem's parameter, n (which is smaller than the length of the input), this algorithm takes time $O(n^3)$ on a RAM, or any related classical machine model with random-access memory.

(1) as $\epsilon \to 0$, $g(\epsilon; a_{IJ}) \to \mathrm{Det}(\ell_{ij}(a_{IJ})) = \mathrm{Perm}(a_{ij})$.
(2) after some algebra one can show that for all but finitely many ϵ (e.g., if R is an algebraically closed field, then for a non-empty Zariski subset of $\mathbb{A}^1(R)$) there is a transformation from $g(\epsilon; a_{ij}$ to a $p(n) \times p(n)$ determinant (where the entries of the transformation matrix and its inverse are—unfortunately—uncontrolled rational functions in ϵ), and hence for such ϵ, $g(\epsilon; a_{IJ})$ exhibits the same symmetries as the $p(n) \times p(n)$ determinant.

The Mulmuley-Sohoni approach to Permanent Versus Determinant can be stated as a two-step approach:

(1) forming g above, it follows that there is an infinitesimal neighbourhood of $g(0; a_{IJ})$ that exhibits the symmetrices of the $n \times n$ determinant, and
(2) study the Zariski closure of all elements of $p(n) \times p(n)$ affine space that exhibit the full (or a large) set of symmetrices of the $n \times n$ determinant, in an attempt to show that the (image of) $Perm(a_{ij})$ is not there.

We are very enthusiastic regarding the first step of the approach. We believe the second step is possible, although it may be better to consider a much smaller symmetry group for the second step. This is suggested to us by an alternate view of this approach; let us explain.

It seems pedagogically simpler to imagine a similar idea on an alternating tree of $+, \cdot$ of depth $d(n)$ (the algebraic analog of an alternating AND, OR tree with literals and their negations at the bottom). In this way you have analogous continuous deformations: for example, if a balanced binary tree with alternating layers of $+, \cdot$ has a $\cdot$ at the bottom, and depth $d = d(n)$, then with 2^d variables $y_1, \ldots, y_{2^d}$, for every integer k one has $y_{2k-1}y_{2k} = (y_{2k-1}s_k)(y_{2k}/s_k)$, giving 2^{d-1} parameters $s_1, \ldots, s_k$ inducing a continuous action on the leaves of the tree leaving the result unchanged. However, one also has the discrete group of symmetries of the full binary tree of depth $d = d(n)$. Hence one might consider this finite group action in conjunction with (or not) the continuous action (the two actions commute). The fact that the determinant size is polynomially equivalent to the size of such a tree will be proven in CPCS 536F, Spring 2022, and is a standard fact in a second course in complexity theory.

Of course, it is not clear that studying a finite group action is any better (or different than) the full group of symmetries of the alternating $+, \cdot$ balanced binary tree or (what is likely equivalent) the full group of symmetries of the determinant.

We believe all directions that involve the Mulmuley-Sohoni idea of exploiting symmetries of the determinant or of a balanced tree are worth pursuing.

It also seems that it would be useful to develop more aspects of singularity theory, with a specific eye to the Mulmuley-Sohoni approach.

At present, the best lower bound on $p(n)$ of which we are aware is quadratic in n.

6.2. **Cohomology Theories for Boolean Functions.** In the late 1970's and early 1980's Michael Ben-Or remarked that the same bounds one uses to lower bound the *algebraic complexity* of functions by counting connected components of level surfaces could be improved by counting the sum of the Betti numbers of these surfaces (in view of the Milnor-Thom bound). Hence there was some enthusiasm for looking for ETC.

6.3. **Both Roads May Lead to SGA4.1.** The main point in the first subsection above is that you may want to learn something about perturbation theory, algebraic groups, etc. The main point in the second subsection is that you may want to study what are called "Grothendieck topologies," or what Grothendieck et al. refer to as a *cite* [sga72]. Hence it seems worth consider some of the remarkable generality developed there, and a concrete example such as Section 5 of Exposé I, which proves the existence of *adjoint functors* which explain how extension by zero works in topological sheaf theory, how induced representations work[7], etc.

The other point is that the set theory described there—Grothendieck et al.'s theory of universes—is especially satisfying. It points out that although humans may perceive things like the real numbers, continua, etc., the only universe one "concretely sees" (this is a human expression, not a precise remark) is the *denumerable universe* of certain finite sets. Hence it seems likely that anything one says about P Versus NP, Determinant Versus Permanent, etc., can presumably be ultimately translated to a proof in terms of finite sets (even if a field is infinite, say the reals or the complex numbers, it seems likely that one can prove the same theorem in a sufficiently large extension of the rationals).

6.4. **Approaches for the Near Future.** It seems that in the near future one would want to further investigate the above approaches further, not only to solve well-studied problems like P Versus NP or Permanent Versus Determinant, but to develop "lower bounds" (e.g., on the size and depth of circuits and formulas) for the complexity of any algebraic or Boolean function.

There is every reason to be optimistic about such investigations, which are compelling regardless of what results they ultimately yield.

It seems like the next steps would be to develop more facts about sheaf and cohomology theory on finite graphs and related structure, and to further develop perturbation and/or singularity theory in a way that we could study the limits of families with a large number of symmetries.

7. Majority Functions

In this section we introduce various of majority functions and their *iterates*, which are intriguing to consider and serve as toy examples for how decisions can be made for various applications.

We will also use them on some of the exercises.

8. Basic Results on Formula/Circuit Size/Depth

The point of this section is to describe some basic facts about formula/circuit size/depth in the Boolean and algebraic settings, some of which are almost completely analogous, and some of which differ between these settings.

9. Tree Symmetries

In this section we study the symmetries of a complete binary tree of a fixed depth d, which is viewed as computing a function of its 2^d leaves, where the operations on the tree are alternating additions and multiplications.

[7] We thank Lior Silberman for this observation.

10. UNDER CONSTRUCTION: EXERCISES

References

[MS01] Ketan Dattatraya Mulmuley and Milind Sohoni, *Geometric complexity theory I: An approach to the P vs. NP and related problems*, SIAM J. Computing **31** (2001), no. 2, 496–526, Subsequent papers available at `http://www.cs.uchicago.edu/people/mulmuley` .

[sga72] *Théorie des topos et cohomologie étale des schémas. Tome 1: Théorie des topos*, Springer-Verlag, Berlin, 1972, Séminaire de Géométrie Algébrique du Bois-Marie 1963–1964 (SGA 4), Dirigé par M. Artin, A. Grothendieck, et J. L. Verdier. Avec la collaboration de N. Bourbaki, P. Deligne et B. Saint-Donat, Lecture Notes in Mathematics, Vol. 269. MR 50 #7130

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`