Entropy and Complexity Lower Bounds

Luc Pellissier
IMERL, Universidad de la República
Julio Herrera y Reissig 565 CP11300.
Montevideo, Uruguay
pellissier@fing.edu.uy

Thomas Seiller
CNRS
LIPN – UMR 7030, Université Paris 13
Sorbonne Paris Cité, France
seiller@lipn.fr

1 Introduction

Lower bounds. The task of *classifying* the complexity classes remains one of the essential open problems in Complexity Theory. As part of the classification problem, researchers have traditionally been concerned with proving *separation results*. Proving that two classes $B \subset A$ are not equal can be reduced to finding lower bounds for problems in A: by proving that certain problems cannot be solved with less than certain resources on a specific model of computation, one can show that two classes are not equal. Conversely, proving a separation result $B \subsetneq A$ provides a lower bound for the problems that are A*-complete* [6] – i.e. problems that are in some way *universal* for the class A.

Alas, the proven lower bound results are very few, and most separation problems remain as generally accepted conjectures. The failure of most techniques of proof has been studied in itself, which lead to the proof of the existence of negative results that are commonly called *barriers*. Altogether, these results show that all proof methods we know are ineffective with respect to proving interesting lower bounds.

Implicit Computational Complexity. An approach to complexity theory that emerged in the recent years is Implicit Computational Complexity (ICC). Related to logical approaches of computational complexity such as Descriptive Complexity, the aim of ICC is to study algorithmic complexity only in terms of restrictions of languages and computational principles. It has been established since Bellantoni and Cook' landmark paper [3], and following work by Leivant and Marion [12, 13].

As part of ICC techniques, some approaches derive from the proofs-as-programs (or Curry–Howard) correspondence. At its core, this correspondence allows one to view the execution of a program as the cut-elimination procedure of a corresponding proof in a formal deductive system (e.g. sequent calculus). Initially stated for intuitionnistic logic [10], the correspondence extends to resource-aware logics such as linear logic (LL), which is well-suited to study computation. This approach to ICC therefore relies on restrictions on the deductive system considered to characterise complexity classes.

Dynamic Semantics. The geometry of interaction program was proposed by Girard [8] shortly after the inception of linear logic. In opposition to traditional denotational semantics – e.g. domains –, the GOI program aims at giving an account of the proofs and programs which also interprets their dynamical features, i.e. cut-elimination/execution. This program is well-suited for tackling problems involving computational complexity, and indeed, geometry of interaction's first model was used to prove the optimality of Lamping's reduction in λ-calculus [9]. More recently, a series of characterisations of complexity classes were obtained using GOI techniques [2, 1].

Among the most recent and full-fledged embodiement of this program lie the second author's Interaction Graphs models [17, 19]. These models, in which proofs/programs are interpreted as *graphings* – generalisations of dynamical systems –, encompass all previous GOI models introduced by Girard [19]. In particular, Interaction Graphs allow for modelling quantitative features of programs/proofs [17].

Submitted to:
TLLA 2018

Semantic Approach to Complexity. Based on a study of several Interaction Graphs models characterising complexity classes [18, 21], the second author has proposed to use graphings to develop a semantic approach to complexity theory [16]. The basic idea behind this program is to model and study programs as dynamical systems that acts on a space – thought of as the space of configurations. As dynamical systems are inherently deterministic, the use of graphings is needed to extend the approach to probabilistic and/or non-deterministic programs. One can then study a program through the geometry of the associated graphing (e.g. a configuration caught in a loop is represented as a point of the space of finite orbit).

The second author conjectures that advanced methods developed within the theory of dynamical systems, in particular methods specific to the study of ergodic theory using techniques from operator algebras, could enable new proof techniques for separation, arguably bypassing barriers [20].

Complexity lower bounds and algebraic geometry. To this day, only one research program is commonly believed to have the ability to bypass all known barriers: Mulmuley's Geometric Complexity Theory (GCT) program [15]. The GCT program aims to prove the PTIME $\neq$ NPTIME lower bound by showing that certain algebraic surfaces[1] cannot be embedded one into the other. Recently, some negative results [11] have closed the easiest path towards proving the conjecture promised by GCT.

Graphings and Lower Bounds. The present work reports on a study of different proofs of lower bounds through the prism of graphings. The authors are convinced that the techniques involved are somehow of a semantic nature and that algebraic geometry does not play the essential role.

This work shows how this point of view allows to recast a number of lower bounds results from the literature. The proofs are understood as a study of the geometry of the graphing interpretation of machines, compared to the geometry of specific problems. This stresses the fact that the use of methods from algebraic geometry is an implicit choice guided by the specific models considered. The abstraction by means of graphings opens the way to applying the techniques to other models of computation, greatly widening the scope of geometric methods for tackling lower bound results.

2 Abstract Programs and Entropy

Abstract Programs. The first step of our reconstruction of the algebraic lower bound proofs is to express programs as graphings. This point of view allow to isolate two components:

- an *abstract model of computation* (AMC) – which we define as a triple (G, R, α), where $\langle G, \mathrm{R}\rangle$ is a presentation of a monoid $\mathrm{M}\langle G, \mathrm{R}\rangle$ and α is a monoid action $\mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ – that represent the different operations a program in a given model can use;
- an *abstract program* within an AMC which is a generalized dynamical system (a graphing) that can use operations of the AMC;

So, for instance, a particular RAM is an abstract program in the AMC of RAMs, while a particular algebraic circuit is an abstract program in the AMC of algebraic circuits.

This approach has a major interest in that it allows to define a purely algebraic operation, related to the notion of *amalgamated free product of groups*, to deal with parallelisation of abstract models of computation: the CREW of AMCs with the *concurrent read, parallel write* discipline. This presentation of parallelism, without reference to the particular model of computation, is, up to our knowledge, a first. It moreover shows how the interpretation of programs as graphings goes beyond the limitation of the Curry-Howard correspondence to functional programs.

[1] One representing the permanent, one the discriminant, functions which have different complexity if PTIME $\neq$ NPTIME [23].

Entropy. We then explain how a graphing induces geometric decompositions of the space it acts on, describing the orbits – the computational traces – of a point through iterations of the graphing. The geometric decomposition obtained after k iterations is called the *k-th cell decomposition* CELL(k) of the space w.r.t. the graphing. It is the coarsest decomposition that has the crucial property that for any two initial configurations contained in the same cell of CELL(k), the k-th first iterations of the graphing on these two configurations go through the same sequence of states.

We generalise the notion of topological entropy, which quantifies the exponential growth of the number of orbits of a dynamical system, to define the entropy of a graphing. We then give bounds on the number of cells of the decomposition.

Theorem 1. *Let G be a deterministic graphing with entropy $h(G)$. The cardinality $c(k)$ of* CELL(k) *is asymptotically bounded by $g(k) = 2^k 2^{h([G])}$, i.e. $c(k) = O(g(k))$.*

In the specific models considered, the decomposition CELL(k) is naturally induced by regions delimited by algebraic varieties. We can refine the above theorem to more specific situations in which both the number and the degrees of the varieties delimiting the regions can be bounded. This is the place where geometric algebraic methods enter into play.

3 Lower Bounds

After that, all that remains to be done is to find a suitable problem for each model and show that its geometry is too complicated for the model at hand. Based on these methods, we consider three distinct models of computation and show how the bounds obtained above generalise three different types of lower bounds proofs from the literature.

Algebraic Decision/Computation Trees. The model of algebraic decision tree is a very simple computational model. Basically, one is given a point $\vec{x}$ in the space $\mathbf{R}^k$, and at each step in the computation one asks whether a given polynomial P cancels at $\vec{x}$, allowing the program to branch depending on the answer. After a finite number of tests, the program outputs either "yes" or "no". Obviously, such a program can be represented as a binary tree whose branches are labelled by polynomials and leaves are labelled by "yes" or "no". An algebraic decision tree T decides a set $W(T)$, namely the subset of $\mathbf{R}^k$ of all points $\vec{x}$ such that the decision tree, given $\vec{x}$ as input, produces a "yes". Steel and Yao [22] showed how the number of connected components of a subset $W \subset \mathbf{R}^k$ provides lower bounds on both the depth of, and the degrees of the polynomials appearing in, any algebraic decision tree T such that $W = W(T)$. This allowed them to provide lower bounds for a number of problems, such as the knapsack problem.

Later work by Ben-Or [4] extends this result to the model of *algebraic computation trees* which perform at each step some algebraic operations on the inputs.

Algebraic NC. The second lower bounds result takes place in *algebraic complexity*, and relates to the BSS model of computation [5]. A BSS model is basically a Turing machine acting on first order structures w.r.t. a given signature (i.e. constants, function symbols and predicates): the tape contains arbitrary elements from the underlying set, and the machine is allowed to perform operations defined by the interpretation of function symbols and tests based on the interpretations of predicates. The most studied such model is the model of computation on the reals, with the sum and product as basic operations, and a single relation corresponding to the identity. Based on this model, one easily defines the notion of polynomial time computation on the reals (w.r.t. this model[2]), and the complexity class $\text{PTIME}_{\mathbf{R}}$. In this line of work, researchers have also considered real analogs of boolean circuits, allowing for the definition of the class

[2] Of course, other definitions of polynomial time for real computation exist.

$\mathrm{NC}_{\mathbf{R}}$. One striking aspect of this approach to real computation is the existence of a separation result between $\mathrm{NC}_{\mathbf{R}}$ and $\mathrm{PTIME}_{\mathbf{R}}$, obtained by Cucker [7].

PRAMs without bit operations. The last result we relate to our method is a lower bound result due to Mulmuley [14]. This result, though twenty years old, is still considered today as one of the strongest known separation result. It shows that some algebraic counterpart of NC, defined through an ad-hoc algebraic model named "PRAMs without bit operations", is strictly included in PTIME. The strength of this result lies in the fact that the ad-hoc complexity class defined through the model of PRAM $^-$, though strong enough to contain usual polynomial-time problems, is shown to be separated from the standard class of polynomial time functions.

Our graphing-based approach allows us to exhibit that in all the above proofs of lower bounds, the methodology is the same. First, one realises the above models as abstract models of computation. This allows one to get upper bounds on the cell decompositions (the number of cells, the number and degrees of the polynomials delimiting them) induced by a machine in the model (a computation tree, an real circuit, a PRAM $^-$) by Theorem 1 and its refinement. Then, one exhibits a problem for which it is possible to compute lower bounds on the cell decomposition needed to compute it (the knapsack problem in Steele and Yao, some polynomial family in the case of algebraic circuits, the MAXFLOW problem in the case of PRAM $^-$), yielding lower bounds in the computational model. For Cucker's and Mulmulmey's results, the chosen problem is shown to be uncomputable by the chosen model; as it is known to belong to a larger class ($\mathrm{PTIME}_{\mathbf{R}}$ for Cucker's result, PTIME for Mulmuley's), this lower bound provides a separation result.

4 The example of PRAM

The AMC of RAMs is defined by generators that modify the state of the memory. We represent memory as two infinite sequence of integers $\mathbf{Z}^\omega \times \mathbf{Z}^\omega$ parametrized by the input, that is functions $\mathbf{Z}^d \to \mathbf{Z}^\omega \times \mathbf{Z}^\omega$: the first copy of $\mathbf{Z}^\omega$ represents shared registers while the second represents private registers. For instance, the generator $\mathtt{plus}(i)$ corresponding to the instruction $\mathtt{X_i} := \mathtt{X_i} + 1$; acts on the configuration space as:

$$f \mapsto P_i \circ f, \text{ with } P_i : (x_1, \dots, x_i, \dots) \mapsto (x_1, \dots, x_{i-1}, x_i + 1, x_{i+1} \dots)$$

while the generator $\mathtt{copy}(i, \sharp j)$ corresponding to the instruction $\mathtt{X_i} := \sharp \mathtt{X_j}$ acts as:

$$f \mapsto CR_{i,j} \circ f, \text{ with } CR_{i,j} : (x_1, \dots, x_i, \dots) \mapsto (x_1, \dots, x_{i-1}, x_{x_j}, x_{i+1} \dots).$$

In this case one obtains the following refinement of Thm. 1.

Theorem 2. *Let G be a deterministic graphing interpreting a* PRAM *with p processors. Then* CELL(k) *is determined by at most* $2^k p^k$ *algebraic varieties whose defining polynomials' degree are bounded by* 2^k.

We show how to associate a partition of a space to a decision problem obtained from a parametrization of an optimization problem, such as the MAXFLOW problem. In a nutshell, given an optimization problem, and a real parametrization f of it (a function that maps reals into valid instances of the problem), the space $\mathbf{Z}^3$ is divided in two subspaces: the subspace of points (x, y, z) such that x/z is lesser than the optimal value of the optimisation problem on $f(y/z)$ and the others.

Although the geometry of the problem is fairly simple, we show that a machine deciding it would need to induce a decomposition in cells of the space whose *complexity*[3] satisfy a certain relation w.r.t. the complexity of the problem, the number of processors, and the computation time. This is the only

[3]The complexity of a decomposition is here measured by the variations of the varieties that delimitates it.

place in all the demonstration where algebraic geometry enter the picture: indeed, on the one hand the geometry of a machine has been encoded as a set of algebraic surfaces, and an algebraic-geometric notion of complexity of the surfaces is expressed in function of the computation times and the number of proccesors; on the other hand, the geometry of the problem is also coded as a set of surfaces (albeit very simple). The comparison between the two involves the Milnor-Thom theorem.

After having defined a variation on the PRAM model allowing to have finer complexity bounds, we finally show Mulmuley's result, i.e. a lower bound for the MAXFLOW problem.

Theorem 3. *Let G be a graphing interpreting a* PRAM *without bit operations with $2^{O(N^c)}$ processors, with N the length of the inputs and c any positive integer. Then G does not decide* MAXFLOW *in $O(N^c)$ steps.*

References

[1] C. Aubert, M. Bagnol & T. Seiller (2016): *Unary Resolution: Characterizing Ptime*. In: *FOSSACS*.

[2] C. Aubert & T. Seiller (2016): *Logarithmic Space and Permutations*. *Inf. Comp.* 248.

[3] S. Bellantoni & S. Cook (1992): *A new recursion-theoretic characterization of the polytime functions*. *Computational Complexity* 2.

[4] M. Ben-Or (1983): *Lower Bounds for Algebraic Computation Trees*. In: *STOC*.

[5] Lenore Blum, Mike Shub & Steve Smale (1989): *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*. *American Mathematical Society. Bulletin. New Series* 21(1), pp. 1–46, doi:10.1090/S0273-0979-1989-15750-9.

[6] S. Cook (1971): *The complexity of theorem-proving procedures*. In: *STOC*.

[7] Felipe Cucker (1992): $\text{PTIME}_{\mathbf{R}} \neq \text{NC}_{\mathbf{R}}$. *Journal of Complexity* 8(3), pp. 230–238, doi:10.1016/0885-064X(92)90024-6.

[8] J.-Y. Girard (1989): *Towards a Geometry of Interaction*. *Contemporary Mathematics* 92.

[9] G. Gonthier, M. Abadi & J.-J. Lévy (1992): *The Geometry of Optimal Lambda Reduction*. In: *Proc. 19th ACM Symposium on Principles of Programming Languages*.

[10] W. A. Howard (1980): *The formulas-as-types notion of construction*. In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*.

[11] C. Ikenmeyer & G. Panova (2017): *Rectangular Kronecker coefficients and plethysms in geometric complexity theory*. *Advances in Mathematics* 319.

[12] D. Leivant & J.-Y. Marion (1993): *Lambda calculus characterizations of poly-time*. *Fundam. Inform.* 19.

[13] S. Leivant & J.-Y. Marion (1994): *Ramified recurrence and computational complexity II: Substitution and poly-space*. *Lecture Notes in Computer Science* 933.

[14] K. Mulmuley (1999): *Lower Bounds in a Parallel Model without Bit Operations*. *SIAM J. Comp.* 28.

[15] K. D. Mulmuley (2012): *The GCT Program Toward the P vs. NP Problem*. *Commun. ACM* 55.

[16] T. Seiller (2015): *Towards a* Complexity-through-Realizability *Theory*. Arxiv:1502.01257.

[17] T. Seiller (2016): *Interaction Graphs: Full Linear Logic*. In: *IEEE/ACM LICS*.

[18] T. Seiller (2016): *Interaction Graphs: Nondeterministic Automata*. Under revision.

[19] T. Seiller (2017): *Interaction Graphs: Graphings*. *Ann. of Pure and Applied Logic* 168.

[20] T. Seiller (2017): *Why Complexity Theorists Should Care About Philosophy*. In J. Fichot & T. Piecha, editors: *Beyond Logic. Proc. of the Conference held in Cerisy-la-Salle*.

[21] T. Seiller (2018): *Interaction Graphs: Probabilistic Automata*. In preparation.

[22] J. M. Steele & A. Yao (1982): *Lower bounds for algebraic decision trees*. *J. Algorithms* 3.

[23] L. G. Valiant (1979): *The complexity of computing the permanent*. *Th. Comp. Sci.* 8.