TOPOLOGICAL BOUNDS ON ALGEBRAIC COMPUTATION TREE COMPLEXITY

ROBBERT LIU

Abstract. In this essay, we explore algebraic computation trees (ACTs), a model of computation which extends the notion of decision problem to languages which are subsets of $\mathbb{R}^n$. We define the algebraic computation tree complexity of a language $\mathcal{L}$, and prove that the ACT complexity of $\mathcal{L}$ has a lower bound which depends on the number of connected components of $\mathcal{L}$ with respect to the topology of $\mathbb{R}^n$. Finally, we use this lower bound to show that any ACT which computes the problem of determining whether n numbers are distinct takes at most $\mathcal{O}(n \log n)$ nontrivial steps.

1. Introduction

The Turing machine models computations involving discrete values, such as bits or integers. However, numerous important algorithms in mathematics require the manipulation of real numbers, or in general, elements of an arbitrary field. This motivates the search for reasonable models of computation enabling arithmetic on structures such as $\mathbb{R}$ or $\mathbb{C}$.

The problem with representing real-valued computations using classical models, such as the Turing machine, is that the complexity of $\mathbb{R}$ far exceeds that of $\mathbb{Z}$. For example, a single real number could encode the answer to every instance of an undecidable problem, such as the halting problem. Thus, the resulting models of computation tend to be more powerful than classical models. We explore one particular model of computation, the **algebraic computation tree**, which generalizes the notion of computable functions to algebraic functions $\mathbb{R}^n \to \{0, 1\}$ or languages $\mathcal{L} \subseteq \mathbb{R}^n$. The proofs and exposition in this essay are mostly adapted from Chapter 16.2 of [AB16].

2. Algebraic Computation Trees

Definition 2.1 (Algebraic computation tree over $\mathbb{R}$)**.** An algebraic computation tree $\mathfrak{T} = (V, E, v_0, X)$ is a directed, rooted, binary tree (V, E, v_0) along with a list of variables $\overline{x} = (x_1, \ldots, x_n)$. Each vertex in V is of exactly one of the following types:

(1) A leaf labelled Accept or Reject.
(2) A **computation vertex** u with out-degree 1, labelled $\ell_u(\overline{x}) = f_1(\overline{x}) \circ f_2(\overline{x})$, where $\circ \in \{+, -, \times, \div, \sqrt{}\}$ and $f_1(\overline{x}), f_2(\overline{x})$ are either projections onto some variable x_i or labels of ancestor vertices.
(3) A **branch vertex** v with out-degree 2, labelled $\ell_v(\overline{x}) := g(\overline{x}) \bowtie 0$, where $\bowtie \in \{\leqslant, \geqslant, =\}$ and $g(\overline{x})$ is either projection onto some variable x_i or a label of an ancestor vertex.

Date: April 22, 2022.

Intuitively, computation vertices represent fixed steps in a series of arithmetic computations, and branch vertices represent decisions which depend on previously computed data. Naturally, $\mathfrak{T}$ computes a function $f_{\mathfrak{T}} : \mathbb{R}^n \to \{0,1\}$ that asks for each vector $\overline{y} \subseteq \mathbb{R}^n$ whether replacing the variables x_i with y_i induces a computation path which terminates at an Accept vertex. We give a proper definition below.

Definition 2.2 (Function/language computed by algebraic computation tree)**.** The **function** $f_{\mathfrak{T}} : \mathbb{R}^n \to \{0,1\}$ **computed by** $\mathfrak{T}$ is defined as follows: $f_{\mathfrak{T}}(\overline{y}) = 1$ if there exists a path $v_0 \rightsquigarrow v_n$ in $\mathfrak{T}$, such that

(1) v_n is labelled Accept.
(2) For $0 < i \leqslant n$, exactly one of the following is true:
   (a) v_{i-1} is a computation vertex, and there exists an edge $v_{i-1} \to v_i$.
   (b) v_{i-1} is a branch vertex, and $\ell_v(\overline{x})$ is false if and only if v_i is the left child of v_{i-1}.

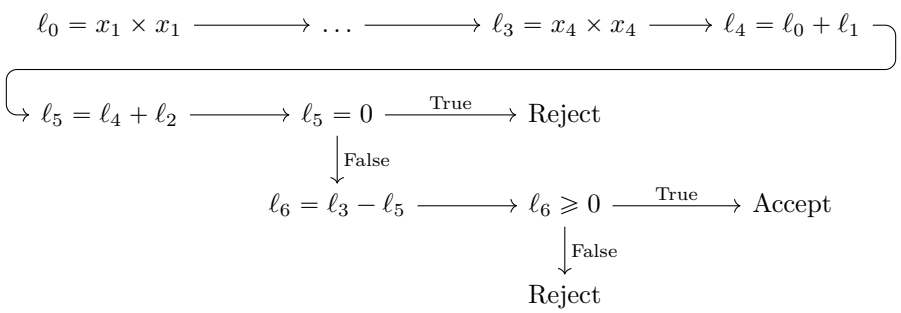
If no such path exists, we define $f_{\mathfrak{T}}(\overline{y}) = 0$. Alternatively, we say that the set $\mathcal{L}_{\mathfrak{T}} \subseteq \mathbb{R}^n$ is the **language computed by** $\mathfrak{T}$ if $f_{\mathfrak{T}}$ is the characteristic function of $\mathcal{L}_{\mathfrak{T}}$. This will be the more useful characterization for us.

ACTs can be generalized to functions on $\mathbb{Q}$ by removing the operation $\sqrt{}$, and furthermore to nonordered fields by only allowing $=$ comparisons.

Definition 2.3 (ACT complexity of a function/language)**.** For any path $p : v_0 \rightsquigarrow v_n$ in $\mathfrak{T}$ from the root to a leaf, let the **cost** of p be the number of computation vertices in p containing the operations $\times, \div, \sqrt{}$ plus the number of branch vertices in p. Let the **depth** of $\mathfrak{T}$ be the maximum cost of any path in it. Define the **ACT** complexity $AC(f)$ of a function $f : \mathbb{R}^n \to \{0,1\}$ (resp. a language $\mathcal{L} \subseteq \mathbb{R}^n$) to be the minimum depth of any ACT that computes f (resp. $\mathcal{L}$).

Hence, this definition arises by assigning a cost of 1 to any vertex representing a nontrivial step.

Example 2.4. We present a depth-6 ACT that computes the language $\{\overline{y} \in \mathbb{R}^4 : 0 < ||(y_1, y_2, y_3)||^2 \leqslant y_4^2\}$, that is, points $\overline{y}$ such that (y_1, y_2, y_3) is contained in the closed ball in $\mathbb{R}^3$ of radius $|y_4|$ punctured at 0.



3. A topological method for lower bounds

In this section, we will show that the ACT complexity of a language $\mathcal{L}$ depends on the topology of $\mathcal{L}$ as a subspace of $\mathbb{R}^n$. $\mathcal{U}$ is called **disconnected** if we can write $\mathcal{U} = U \sqcup V$ as the disjoint union of nonempty open sets U, V; in this case, we call $U \sqcup V$ a **separation** of $\mathcal{U}$. Otherwise, $\mathcal{U}$ is **connected**. A nonempty open subspace

$C \subseteq \mathcal{U}$ is called a **connected component** if it is connected and its complement is open. Let $\#\mathcal{U}$ denote the number of connected components of $\mathcal{U}$, which is possibly infinite. Our goal is to prove the following lower bound on the ACT complexity of a language:

Theorem 3.1 ([BO83])**.** *For any language $\mathcal{L} \subseteq \mathbb{R}^n$,*

$$AC(\mathcal{L}) \in \Omega(\log(\max(\#\mathcal{L}, \#\mathcal{L}^c)) - n).$$

Lemma 3.2. *Recall that f is continuous if $f^{-1}(V)$ is open for any open set $V \subseteq \mathcal{V}$. If $f : \mathcal{U} \to \mathcal{V}$ is a continuous function between topological spaces, then $\#\mathcal{U} \geqslant \#f(\mathcal{U})$.*

Proof. We first prove two simple facts about connected sets. We can easily show that the continuous image of a connected set is connected by contraposition: if $U \sqcup V$ is a separation of $f(\mathcal{U})$, then $f^{-1}(U) \sqcup f^{-1}(V) = f^{-1}(U \sqcup V) = \mathcal{U}$ is a separation of $\mathcal{U}$.

Moreover, the union $\bigcup \alpha \in A U_\alpha$ of connected sets U_α that intersect at a point x is connected, which is also proven using contraposition: if $U \sqcup V$ is a separation of $\bigcup \alpha \in A U_\alpha$, then assume without loss of generality that $x \in U$. Since V is nonempty, it must intersect some set U_β. $U \cap U_\beta \ni x$ is also nonempty, so $(V \cap U_\beta) \sqcup (U \cap U_\beta)$ is a separation of U_β.

We finally turn to our initial statement. If $\mathcal{C}$ is the set of connected components of $\mathcal{U}$, $\mathcal{D} := \{f(C) : C \in \mathcal{C}\}$ is a set of open sets which cover $f(\mathcal{U})$. Then $\mathcal{D}_x := \bigcup_{D \in \mathcal{D}, x \in D} D$ is a connected component of $f(\mathcal{U})$, since $\mathcal{D}_x$ is open, nonempty, connected (by above), and has the open complement $\bigcup_{D \in \mathcal{D}, x \notin D} D$. Thus, $\{\mathcal{D}_x : x \in f(\mathcal{U})\}$ contains the connected components of $f(\mathcal{U})$, and we have

$$\#f(\mathcal{U}) = |\{\mathcal{D}_x : x \in f(\mathcal{U})\}| \leqslant |\mathcal{D}| \leqslant |\mathcal{C}| = \#\mathcal{U}$$

as required. □

This is essentially all of the necessary topology to prove the statement. The next theorem shows that the leaves of an ACT parametrize a family of systems of polynomial constraints, whose solution sets partition $\mathbb{R}^n$.

Lemma 3.3 ([AB16])**.** *If $\mathcal{L}$ has an ACT $\mathfrak{T}$ of depth d, then $\mathcal{L}$ (and $\mathcal{L}^c$) is a union of at most 2^d sets $C_1, C_2, \cdots \subseteq \mathbb{R}^n$, where C_i characterized by the following property: there are up to d equations of the form $p_{i_r}(y_1, \ldots, y_d, x_1, \ldots, x_n) \bowtie 0$, such that p_{i_r} is a polynomial with degree $\leqslant 2$ for $r \leqslant d$ and $\bowtie \in \{\leqslant, \geqslant, =, \neq\}$. Then, C_i is the set of points $\overline{x} \in \mathbb{R}^n$ which admit some $\overline{y} \in \mathbb{R}^d$ such that $(\overline{x}, \overline{y})$ is a solution to the system above. Additionally, if we allow twice as many y_i's, we may assume without loss of generality that there are no $\neq$ comparisons in the systems of constraints.*

Proof. This is a matter of translating the definition of ACT into the conclusion of this theorem. Each C_l is associated to a leaf l of $\mathfrak{T}$ and consists of the points $\overline{x}$ which end up at that leaf when computed by $\mathfrak{T}$. The polynomials p_{l_r} can be derived from the labels of the vertices in the path $p : v_0 \rightsquigarrow l$. For example, we associate to a computation vertex labelled $\ell_u(\overline{x}) = \ell_v(\overline{x}) \div x_i$ the polynomial $y_v x_i - y_v = 0$, and we associate to a branch vertex labelled $\ell_w(\overline{x}) \leqslant 0$ the polynomial $y_w \leqslant 0$. Since $\mathfrak{T}$ has depth d, it follows that p has at most d vertices, which induce at most d polynomials p_{i_r}. Finally, $\mathfrak{T}$ has at most 2^d leaves. The details are fleshed out in the proof of lemma 16.21 in [AB16].

To transform a constraint of the form $p_i(\overline{y}, \overline{x}) \neq 0$ into one without $\neq$, we can instead use the constraint $1 - z_i p_i(\overline{y}, \overline{x}) = 0$ with a newly introduced variable z_i,

since any non-zero element is invertible in a field. This is called **Rabinovotch's trick**. Similarly, we can replace $p_i(\overline{y}, \overline{x}) > 0$ with $p_i(\overline{y}, \overline{x}) - z_i^2 = 0$. □

Next, we mention a result from algebraic geometry which gives a lower bound on the number of connected components of a solution set to a system of constraints. This is, in a sense, the central cog in Theorem 3.1.

Theorem 3.4 (Milnor-Thom [AB16])**.** *If $S \subseteq \mathbb{R}^n$ is defined by degree d constraints with m inequalities and h equalities, then*

$$\#(S) \leqslant d(2d-1)^{n+h-1}.$$

Proof of Theorem 3.1. Let $\mathfrak{T}$ be an ACT of depth d that computes $\mathcal{L}$. $\mathfrak{T}$ has at most 2^d leaves, each leaf l admitting a set C_l of constraints $p_{l_i} = (\overline{y}, \overline{x})$ of degree 2, where $(\overline{y}, \overline{x}) \subseteq \mathbb{R}^{d+n}$. Let $S_l \subseteq \mathbb{R}^{d+n}$ be the set of solutions $(\overline{y}, \overline{x})$ to C_l. If $\mathcal{L}_l \subseteq \mathbb{R}^n$ is the set of inputs $\overline{x}$ which reach l when computed by $\mathfrak{T}$, then $\mathcal{L}_l$ is the projection of C_l onto the last n coordinates $\overline{x}$. Since projection is continuous, Lemma 3.2 implies $\#\mathcal{L}_l \leqslant \#C_l$. By Theorem 3.4, $\#C_l \leqslant 2 \cdot 3^{n+d-1} \leqslant 3^{n+d}$ so $\mathcal{L}$ has at most $2^d 3^{n+d}$ connected components, allowing us to conclude $d \geqslant \Omega(\log(\#\mathcal{L})) - \mathcal{O}(n)$. We can repeat this proof with $\mathcal{L}^c$ instead to get $d \geqslant \Omega(\log(\#\mathcal{L}^c)) - \mathcal{O}(n)$. □

4. An application

We can use ACTs to study the complexity of real-valued analogs of common decision problems. Let Element Distinctness denote the problem of deciding whether n input numbers $x_1, \ldots, x_n$ are all distinct. The naive algorithm computes solves this problem in at $\mathcal{O}(n \log n)$ steps: sort the numbers in $\mathcal{O}(n \log n)$ steps and compare each pair of adjacent numbers in $\mathcal{O}(n)$ steps. It turns out that this algorithm is asymptotically optimal in the ACT model.

Theorem 4.1 (Lower bound for the ACT complexity of Element Distinctness [AB16])**.** *Let $\mathcal{L} = \{\overline{x} \in \mathbb{R}^n : \prod_{i<j}(x_i - x_j) \neq 0\}$, which is an equivalent characterization of Element Distinctness. Then, $\#\mathcal{L} \geqslant n!$, so Theorem 3.1 implies that $AC(\mathcal{L}) \in \Omega(\log(n!) - n) = \Omega(n \log n)$.*

Proof. For an n-permutation σ, define $\mathcal{L}_\sigma = \{\overline{x} \in \mathcal{L} : x_{\sigma(1)} < \cdots < x_{\sigma(n)}\}$. Since there are $n!$ distinct n-permutations, it suffices to show that $\mathcal{L}_\sigma$ and $\mathcal{L}_{\sigma'}$ are not subsets of a single connected component of $\mathcal{L}$ for $\sigma \neq \sigma'$.

For any distinct permutations σ, σ', we have some i, j such that $\sigma^{-1}(i) < \sigma^{-1}(j)$ but $(\sigma')^{-1}(i) > (\sigma')^{-1}(j)$. Thus, all points $\overline{x} \in \sigma$ satisfy $x_i - x_j > 0$, whereas points $\overline{x} \in \sigma$ satisfy $x_i - x_j < 0$. If we suppose for contradiction that a single connected component $\mathcal{C} \subseteq \mathcal{L}$ contains $\mathcal{L}_\sigma$ and $\mathcal{L}_{\sigma'}$, then the function $f : \mathcal{C} \to \mathbb{R}$ defined by $\overline{x} = x_i - x_j$ attains both a negative and positive value. Since $\mathcal{C}$ is connected and f is continuous, the intermediate value theorem implies that $f(\overline{y}) = 0$ for some $\overline{y} \in \mathcal{L}$. Hence $y_i = y_j$, which contradicts the assumption that $\overline{y}$ is a list of n distinct numbers. □

References

[AB16] Sanjeev Arora and Boaz Barak, *Computational complexity: a modern approach*, Cambridge University Press, 2016. 1, 3, 4

[BO83] Michael Ben-Or, *Lower bounds for algebraic computation trees*, Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '83, Association for Computing Machinery, 1983, pp. 80–86. 3