Algebraic Computation Models

Ζυγομήτρος Ευάγγελος
ΜΠΛΑ 201118

Φεβρουάριος, 2013

Reasons for using algebraic models of computation

- The Turing machine model captures computations on bits. Many algorithms are most naturally described as operating on uncountable sets, like $\mathbb{R}$ or $\mathbb{C}$ (e.g. Newton's method for finding roots of a given real-valued function f).
- A wide variety of such algorithms occurs in numerical analysis, computer algebra, computational geometry and robotics, typically assuming that a basic computational step involves an operation $(+, \times, /)$ in some arbitrary field $\mathbb{F}$.
- Allowing arbitrary field operations in an algorithm may not be directly implentable (since computers do arithmetic using finite precision) but it provides a useful approximation to the asymptotic behavior of the algorithm, as computers are allowed to use more and more precision in their computations.

Pitfalls

Allowing (arbitrary precision) arithmetic on real numbers as a basic step can quickly lead to unrealistically strong models.

Example

Shamir has shown how to factor any integer N in $poly(logN)$ time on any model that allows arithmetic (including the mod operation) with arbitrary precision (whereas factoring is a notoriously hard problem for classical TMs).

Example

A real number can encode infinite amount of information, e.g. a single real number is enough to encode the answer to every instance of SAT. Thus, we have to be careful in defining a model that allows even a single hardwired real number in its programs.

The usual way to avoid such pitfalls is to restrict the algorithm's ability to access individual bits.

Models

- Algebraic Straight-Line Programs
- Algebraic Circuits
- Algebraic Turing Machines

We will consider algorithms that get as input a tuple of numbers over a field or a ring $\mathbb{F}$ (typically $\mathbb{R}$ or $\mathbb{C}$). The input $(x_1, x_2, \ldots, x_n) \in \mathbb{F}^n$ is said to have size n. A language over a field/ring $\mathbb{F}$ is a subset of $\cup_{n \geq 1} \mathbb{F}^n$.

Algebraic Straight-Line Programs

Definition (Algebraic straight-line program over $\mathbb{F}$)

An algebraic straight-line program of length T with input variables $x_1, x_2, \ldots, x_n \in \mathbb{F}$ and built-in constants $c_1, c_2, \ldots, c_n \in \mathbb{F}$ is a sequence of T statements of the form

$$y_i = z_{i1} \star z_{i2}, \quad i = 1, 2, \ldots, T$$

where $\star$ is one of the field operations $+$ or $\times$ and each of z_{i1}, z_{i2} is either an input variable, or a built-in constant, or y_j for $j < i$.

The straight-line computation consists of executing these simple statements in order, finding values for $y_1, y_2, \ldots, y_T$. The output of the computation is the value of y_T.

- straight-line: no conditionals or loops

Algebraic Straight-Line Programs (cnt'd)

Example (Computation of $e \times (x_1 + e) + \pi \times x_2$ in $\mathbb{R}$)

Input: x_1, x_2
Output: y_4
Built-in constants: π, e
$y_1 = x_1 + e$
$y_2 = e \times y_1$
$y_3 = \pi \times x_2$
$y_4 = y_2 + y_3$

Reminder: The degree of a multivariate polynomial $p(x_1, x_2, \ldots, x_n)$ is defined to be the maximum degree among all its monomials.
The degree of the monomial $c \prod_i x_i^{d_i}$ is $\sum_i d_i$.

Lemma

The output of a staight-line program of length T with variables $x_1, x_2, \ldots, x_n$ is a polynomial $p(x_1, x_2, \ldots, x_n)$ of degree at most 2^T.

Algebraic Straight-Line Programs (cnt'd)

We are interested in asymptotic complexity i.e. the length (as a function of n) of such a program that computes a function f_n of n variables.
Here are some examples of interesting functions that are computable by polynomial length algebraic straight-line programs:

- Polynomial Multiplication:
  - $O(n^2)$ using a naive algorithm
  - $O(n \log n)$ using the fast Fourier transform for fields that have a primitive mth root of unity, where m is the smallest power of 2 greater than $2n$
  - $O(n \log n \log \log n)$ for all fields
- Fast Fourier Transform: $O(n \log n)$ [Cooley and Turkey]
- Matrix Multiplication: $O(n^3)$ (naive algorithm). Improvements using techniques like Strassen's ($O(n^{2.807})$ (1969)) with complexity $O(n^\omega)$ for $\omega < 3$ (current record: $\omega \sim 2.3727$ [Vassilevska Williams] (2011), previous record: $\omega \sim 2.3736$ [Andrew Stothers] (2010) (broke the famous bound, $\omega \sim 2.376$ [Coppersmith and Winograd] (1989))

Algebraic Circuits

Definition

An algebraic circuit consists of an acyclic graph. The leaves are called input nodes, are labeled $x_1, x_2 \ldots x_n$ and take values in a field $\mathbb{F}$. We also allow special input nodes labeled with arbitrary constants $c_1, c_2, \ldots c_k \in \mathbb{F}$. Each internal node, called a gate, is labeled with one of the operations $+, \times$. We consider only circuits with a single output node and with the in-degree of each gate being 2.
An algebraic formula is a circuit where each gate has out-degree equal to 1.

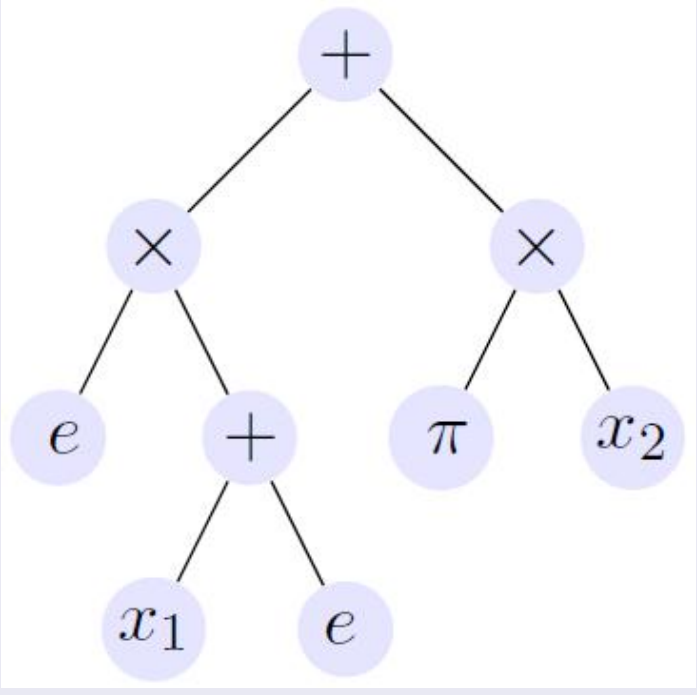
The size of a circuit is the number of gates in it. The depth of the circuit is the length of the longest path from input to output in it.

Lemma

Let $f : \mathbb{F}^n \to \mathbb{F}$ be some function. If f has an algebraic straight-line program of size S, then it has an algebraic circuit of size $3S$. If f is computable by an algebraic circuit of size S then it is computable by an algebraic straight line program of length S.

Algebraic Circuits (cnt'd)

Example (Computation of $e \times (x_1 + e) + \pi \times x_2$ in $\mathbb{R}$)



Input: x_1, x_2
Constants: π, e

Algebraic Circuits (cnt'd)

Definition (Analogs of P and NP for algebraic circuits)

Let $\mathbb{F}$ be a field. We say that a family of polynomials $\{p_n\}_{n\in\mathbb{N}}$, where p_n takes n variables over $\mathbb{F}$, has polynomially-bounded degree if there is a constant c s.t. for every n the degree of p_n is at most cn^c.
The class $\mathbf{AlgP}_{/\mathbf{poly}}$ contains all polynomially bounded degree families of polynomials that are computable by algebraic circuits of polynomial size and polynomial degree.
The class $\mathbf{AlgNP}_{/\mathbf{poly}}$ is the class of polynomially bounded degree families $\{p_n\}$ that are definable as

$$p_n(x_1, x_2, \ldots, x_n) = \sum_{e\in\{0,1\}^{m-n}} g_m(x_1, x_2, \ldots, x_n, e_{n+1}, \ldots, e_m)$$

where $g_m \in \mathbf{AlgP}_{/\mathbf{poly}}$ and m is polynomial in n.

Algebraic Circuits (cnt'd)

Definition

A function $f(x_1, x_2, \ldots, x_n)$ *is a projection of a function* $g(y_1, y_2, \ldots, y_n)$ *if there is a mapping* σ *from* $\{y_1, y_2, \ldots, y_m\}$ *to* $\{0, 1, x_1, x_2, \ldots, x_n\}$ *s.t.* $f(x_1, x_2, \ldots, x_n) = g(\sigma(y_1), \sigma(y_2), \ldots \sigma(y_m))$.
We say that f *is projection-reducible to* g *if* f *is a projection of* g.

Example

The function $f(x_1, x_2) = x_1 + x_2$ is projection reducible to $g(y_1, y_2, y_3) = y_1^2 y_3 + y_2$ since $f(x_1, x_2) = g(1, x_1, x_2)$.

Algebraic Circuits (cnt'd)

Theorem (Completeness of determinant and permanent)

For every field $\mathbb{F}$ and every polynomial family on n variables that is computable by an algebraic formula of size u is projection reducible to the determinant function (over the same field) on $u + 2$ variables.
For every field except those that have characteristic 2, every polynomial family in $\mathbf{AlgNP}_{/\mathbf{poly}}$ is projection reducible to the permanent function (over the same field) with polynomially more variables.

- determinant is $\mathbf{AlgP}_{/\mathbf{poly}}$-complete
- permanent is $\mathbf{AlgNP}_{/\mathbf{poly}}$-complete

It is necessary to show $\mathbf{AlgP}_{/\mathbf{poly}} \neq \mathbf{AlgNP}_{/\mathbf{poly}}$ before one can show $\mathbf{P} \neq \mathbf{NP}$.

The Blum-Shub-Smale Model

- The first uniform model we encounter
- A generalization of Turing Machines
  - input: a string in $\mathbb{F}^n$
  - output: accept or reject
  - each cell can hold an element of $\mathbb{F}$
- The machine has a finite set of internal states. Three categories of states:
  - shift state: move the head to the left or to the right of the current position
  - branch state: if the content of the current cell is a, then goto state q_1 else goto state q_2
  - computation state: replace the content a of the current cell with $f(a)$, where f is a hard-wired function (polynomial or rational depending on whether $\mathbb{F}$ is a ring or a field)

The Blum-Shub-Smale Model (cnt'd)

- In the standard model of the TM, the computation and branch operations can be executed in the same step, whereas here they have to be performed seperately.
- In order to branch, the machine has to be able to remember the value it just read one step ago. For this reason the machine has a single register onto which it can copy the contents of the cell currently under the head, and whose value can be used in the next step.
- Very powerful model (e.g. can compute x^{2^n} in n steps (by repeating the operation $x \leftarrow x^2$)
- The machine can only branch using tests like "Is the content of this cell equal to a?"
- If we allow tests like "Is the content of this cell greater than a?" it would give the machine much more power, including the ability to decide every language in $\mathbf{P}_{/\mathbf{poly}}$ in polynomial time (even if the language is undecidable)
- If we allow rounding as a basic operation then it is possible to factor integers in polynomial time

The Blum-Shub-Smale Model (cnt'd)

Since the BSS model is more powerful than the ordinary Turing Machine, it makes sense to also revisit decidability questions.

Definition (Mandelbrot set decision problem)

For complex c, z, let $p_c(z) = z^2 + c$. Then the Mandelbrot set is defined as

$$\mathcal{M} = \{c \in \mathbb{C} \mid \text{the sequence } p_c(0), p_c(p_c(0)), p_c(p_c(p_c(0))) \ldots \text{ is bounded}\}$$

For example, letting $c = 1$ gives the sequence $0, 1, 2, 5, 26 \ldots$, which tends to infinity. As this sequence is unbounded, 1 is not an element of the Mandelbrot set. On the other hand, $c = i$ gives the sequence $0, i, (-1 + i), -i, (-1 + i), -i, \ldots$, which is bounded, and so i belongs to the Mandelbrot set.

Theorem

$\mathcal{M}$ is undecidable by a machine over $\mathbb{C}$.

Roger Penrose's criticism of artificial intelligence

- Penrose uses a variant of Turing's halting problem to demonstrate that a system can be deterministic without being algorithmic: imagine a system with only 2 states, ON and OFF. If the system's state is ON if a given Turing machine halts, and OFF if the Turing machine does not halt, then the system's state is completely determined by the Turing machine, however there is no algorithmic way to determine whether the Turing machine stops.
- He argues that humans have an intuitive grasp of many things that seem beyond the capacities of the Turing machine model:
  - the present computer is unable to have intelligence because it is an algorithmically deterministic system.
  - the rational processes of the mind are not completely algorithmic and thus cannot be duplicated by a sufficiently complex computer.

  This contrasts with supporters of strong artificial intelligence, who contend that thought can be simulated algorithmically.
- He mentioned the Mandelbrot set as an example. He suggested that such mathematical objects are beyond the purview of computer science, one cannot talk about the decidability of such sets.

Bibliography

- S. Arora and B. Barak, Computational complexity: a modern approach, Cambridge University Press, 2009.
- Salil Vadhan, CS 221: Computational Complexity (Harvard), Lecture Notes (spring 2010).
- R. Penrose, The Emperor's New Mind, Oxford University Press, 1989.