Computer Algebra

Wieb Bosma

Radboud Universiteit Nijmegen

Spring 2010

PART I

INTRODUCTION

What is Computer Algebra?

- No generally accepted definition
  - Algorithms for algebraic objects
  - Exact vs Approximative
  - Symbolic vs Numerical Computing
- This course:
  - Algorithms central
  - Practical usage in mind: complexity!
- In summary:
  - *what* can be computed with modern computer algebra systems, and
  - *how* is it done?

Computational domains

Rough outline of the scope:

> algorithms to compute with combinatorial objects (like graphs), and in those groups, rings, fields, and their associated modules, algebras, etc. for which the objects can be represented and tested for equality on a computer, and for which the operations can be performed effectively.

$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Q}(\alpha)$, $\mathbb{Q}_p$

$\mathbb{Z}/n\mathbb{Z}$, $\mathbb{F}_p$, $\mathbb{F}_q$

$R[x]$, $R[x]/(f)$, $R(x)$,

$R[x_1, x_2, \ldots, x_n]$, $R[x_1, x_2, \ldots, x_n]/I$,

$\mathrm{Hom}(V, W)$, $R[[x]]$, $R((x))$

$\mathrm{Sym}(n)$, K_n

Representation of Objects

Objects stored as a finite number of *bits*. The *size* of an object is the number of bits. Objects may have several distinct *representations*, between which we may have to do *conversions*. But within a fixed representation, objects may have more than one representation: a *normal form* is desirable.

For example:
integers in g-adic representation, or fully factored in primes
polynomials dense (coefficient vectors) or sparse (coefficient, exponent pairs)
permutations cycles, image lists, products of transpositions

Computational tasks

- *Perform the arithmetic operations in the computational domains*

  Addition, multiplication, inversion, powering, composition, actions

- *Normal form computation, conversion between representations*

  Basis representation, factorization

- *Membership and equality testing*

  Conversion, comparison

- *Structural computation, mappings*

  Generators and relations

Many of the most important tasks can be interpreted as conversion between representations!

Computational models

Tasks are executed by way of *algorithms* on *multitape Turing machine* operating on strings of bits. *Computational complexity* is measured in number of *bit operations*.

Sometimes we express operations in a higher level *algebraic model* of computation, where steps are *elementary algebraic operations*.

Example Multiplication of $f, g \in R[x]$, where $\deg f = m$ and $\deg g = n$ can be done with $(m+1)(n+1)$ multiplications and $m \cdot n$ additions in R.

Note that the complexity depends on the representation!

Asymptotics

Complexity functions: partial $f : \mathbb{R} \to \mathbb{R} \cup \infty$ that is defined and non-negative for all integers $n \geq N$.

$f = \mathcal{O}(g)$:

$$\exists C > 0, N \in \mathbb{N} : \forall x > N : f(x) \leq C \cdot g(x)$$

$f = \Omega(g)$:

$$\exists C > 0, N \in \mathbb{N} : \forall x > N : f(x) \geq C \cdot g(x)$$

$f = \Theta(g)$:

$$f = \mathcal{O}(g) \quad \text{and} \quad f = \Omega(g)$$

$f = o(g)$:

$$f(n)/g(n) \to 0 \quad \text{when} \quad n \to \infty$$

Complexity classes

P

the class of algorithms with deterministic polynomial time complexity: the complexity is $\mathcal{O}(x^d)$ for some $d \in \mathbb{N}$

NP

the class of algorithms with non-deterministic polynomial time complexity: the complexity of *verifying* the correctness of a solution (provided by some oracle, say) is $\mathcal{O}(x^d)$ for some $d \in \mathbb{N}$; *finding* the correct solution may not be possible in polynomial time

Sometimes trade-offs between time and space complexity

The true picture of easy versus hard problems may be much more complicated!

Some general techniques

- *probabilistic* rather than deterministic methods (gives to *expected* running times)

  Ex: Pollard ρ algorithm (below)

- iterative and *recursive* methods: divide and conquer

  Ex: Karatsuba algorithm (exercise)

- *homomorphism* methods: mapping to easier structure combined with bounds

  Ex: modular methods (polynomial factorization)

- rewriting

  Ex: Gröbner basis algorithm

Elementary Algorithms

- integer addition and subtraction in $\mathcal{O}(\log n)$
- integer multiplication and division in $\mathcal{O}((\log n)^2)$
- exponentiation (powering) by repeated squaring and multiplication
- polynomial evaluation: Horner's method

First Example: Pollard-ρ

Pollard's ρ method for integer factorization is based on the 'birthday paradox':

taking a random sample of size $\mathcal{O}(\sqrt{n})$ of a set of cardinality n is expected to give a collision

choosing a random $f : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ and x_1, then $x_1, x_2 = f(x_1), x_3 = f(f(x_1)) = f(x_2), \ldots$ mod p will behave randomly in $\mathbb{Z}/p\mathbb{Z}$, for any prime divisor p of n. Hence a collision $x_i \equiv x_j \bmod p$ is expected after $O(\sqrt{p})$ steps! Since $x_i \equiv x_j \bmod N$ is unlikely (especially if $p \ll n$), we detect the unknown p by computing $\gcd(x_i - x_j, n)$.

$f(x) = x^2 + 1 \bmod n$, with $x_1 = 2$ is standard

An optimisation

By the pigeonhole principle the sequence mod p will become periodic after say $s+t$ steps, so that $x_{s+1} \equiv x_{s+t+1} \bmod p$, and the '$\rho$' has a 'tail' of length s and a 'cycle' of length t.

Instead of comparing any two entries x_i, x_j, the same result can be achieved more efficiently by noting that for some m one gets $x_{2m} \equiv x_m \bmod p$, the least such m being the smallest multiple of t exceeding s.

Pollard's ρ method runs heuristically in expected time essentially $\sqrt{p}$ to find the prime factor p

An Overview of Algorithms to Follow

- The Fast Fourier Transform

  and consequences for fast multiplication
- The Euclidean Algorithm

  in many incarnations with many applications
- The Gaussian Elimination Algorithm

  for solving systems, finding matrix inverses and determinants
- The Lenstra-Lenstra-Lovász algorithm

  with surprising applications for short vectors
- Hensel and Newton iteration

  and other methods for root isolation, separation etc.
- Gröbner Basis Algorithm

  again with various applications
- (towards) the Risch Algorithm