Lecture 25: Conclusion

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

April 14, 2021

Overview

- Administrivia
- Foundations of Symbolic Computation
- Computational Linear Algebra
- Modern Computational Algebra
- Computational Invariant Theory
- Topics I wish I had time to cover

Rate this course!

Please log in to

https://evaluate.uwaterloo.ca/

Today is the **last day** to provide us (and the school) with your evaluation and feedback on the course!

- This would really help me figuring out what worked and what didn't for the course
- And let the school know if I was a good boy this term!
- Teaching this course is also a learning experience for me :)

- Administrivia
- Foundations of Symbolic Computation
- Computational Linear Algebra
- Modern Computational Algebra
- Computational Invariant Theory
- Topics I wish I had time to cover

Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:

Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:
  1. Dense representation

$p(x,y,z) \in \mathbb{F}[x,y,z]$ $\deg(p) = 2$

$\updownarrow$

ALL coefficients (include zero coefficients)

$(p_{000}, p_{100}, p_{010}, p_{001}, p_{110}, p_{101}, p_{011}, p_{200}, p_{020}, p_{002})$

$\prod_{i=1}^{n} (x_i + 1)$

2^n terms in dense representation

p_{ijk} = coefficient of $x^i y^j z^k$

Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:
  1. Dense representation
  2. Sparse representation

only tell non-zero coefficients and their corresponding monomials

↕

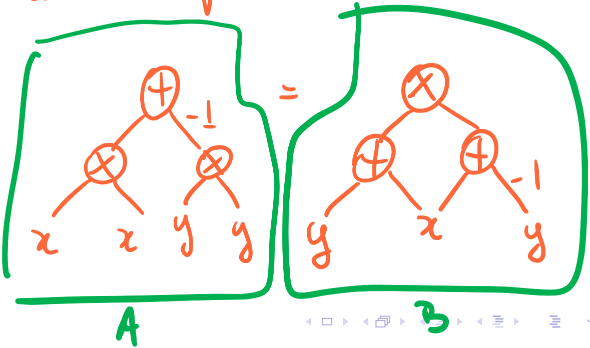
non-zero terms

Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:
  1. Dense representation
  2. Sparse representation
  3. Algebraic circuits (straight-line programs)

represent polynomial by the computations needed to compute it

$x^2 - y^2$



Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:
  1. Dense representation
  2. Sparse representation
  3. Algebraic circuits (straight-line programs)
  4. Black-Box matrix representations

$$\begin{bmatrix} a & b & c \\ d & a & b \\ e & d & a \end{bmatrix} \longleftrightarrow (e, d, a, b, c)$$

Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:
  1. Dense representation
  2. Sparse representation
  3. Algebraic circuits (straight-line programs)
  4. Black-Box matrix representations
- Complexity of certain problems become vastly different depending on representation!

Models of Computation

- In addition to the standard bit representation of integers, we learned *different models* to represent algebraic objects:
  1. Dense representation
  2. Sparse representation
  3. Algebraic circuits (straight-line programs)
  4. Black-Box matrix representations
- Complexity of certain problems become vastly different depending on representation!
- Some open problems:
  1. factoring sparse (univariate or multivariate) polynomials fast
  2. factoring multivariate polynomials computed by algebraic circuits (without restriction on degree)
  3. testing whether two objects from the same model compute the same object

     Given two straight-line programs, do they compute the same polynomial?
  4. more generally, the more succinct the representation, the harder it should be to efficiently solve problems

Fundamental Operations - Multiplication

- Learned how to multiply integers faster

Fundamental Operations - Multiplication

- Learned how to multiply integers faster
- Learned how to multiply polynomials much faster!
  1. Highly non-trivial algorithms!
  2. Karatsuba: reduce number of multiplications needed!

Fundamental Operations - Multiplication

- Learned how to multiply integers faster
- Learned how to multiply polynomials much faster!
  1. Highly non-trivial algorithms!
  2. Karatsuba: reduce number of multiplications needed!
  3. Fast multiplication via interpolation and polynomial evaluation

     Quite far from the intuitive algorithm!
  4. DFT for the rescue!

Fundamental Operations - Multiplication

- Learned how to multiply integers faster
- Learned how to multiply polynomials much faster!
  1. Highly non-trivial algorithms!
  2. Karatsuba: reduce number of multiplications needed!
  3. Fast multiplication via interpolation and polynomial evaluation

     Quite far from the intuitive algorithm!
  4. DFT for the rescue!
- Polynomial multiplication quite often used in practice!

Fundamental Operations - Multiplication

- Learned how to multiply integers faster
- Learned how to multiply polynomials much faster!
  1. Highly non-trivial algorithms!
  2. Karatsuba: reduce number of multiplications needed!
  3. Fast multiplication via interpolation and polynomial evaluation

     Quite far from the intuitive algorithm!
  4. DFT for the rescue!
- Polynomial multiplication quite often used in practice!
- Even more ubiquitous is the Discrete Fourier Transform!
  - Used in audio and video compression [von zur Gathen, Gerhard 2013, Chapter 13] and references
  - many more applications!

Fundamental Operations - Euclidean Algorithm

- Learned how to compute the GCD between integers and two polynomials over $\mathbb{F}[x]$

Fundamental Operations - Euclidean Algorithm

$$af + bg = \gcd(f, g)$$

- Learned how to compute the GCD between integers and two polynomials
- Extended Euclidean Algorithm fundamental for many other problems
  1. Compute inverses in modular computations
  2. Solving Pade Approximation problem in power series approximations

Much simpler to compute linear recurrence sequences!

approximate power series by rational functions

Modular Computation and Chinese Remandering Theorem

- Often times computations over $\mathbb{Z}$ can lead to large intermediate coefficients

Modular Computation and Chinese Remandering Theorem

- Often times computations over $\mathbb{Z}$ can lead to large intermediate coefficients
- Chinese Remaindering Theorem allows us to
  1. "Parallelize" the problem: compute many instances of the problem modulo small primes small coefficients

     Can compute all these instances in parallel!
  2. Control intermediate coefficients

CRT : most of the time we only need to work with small coefficients

Modular Computation and Chinese Remandering Theorem

- Often times computations over $\mathbb{Z}$ can lead to large intermediate coefficients
- Chinese Remaindering Theorem allows us to
  1. "Parallelize" the problem: compute many instances of the problem modulo small primes

     Can compute all these instances in parallel!
  2. Control intermediate coefficients
- It can also be used to give fastest known algorithm for univariate polynomial factoring over finite fields! **Kedlaya-Umans 2011**

Resultants and Polynomial GCD

- GCD between two polynomials can be characterized by *algebraic invariant*: *Resultant*

Resultants and Polynomial GCD

- GCD between two polynomials can be characterized by *algebraic invariant*: *Resultant*
- Use resultant to compute a modular GCD algorithm for two polynomials over $\mathbb{Z}[x]$

Resultants and Polynomial GCD

- GCD between two polynomials can be characterized by *algebraic invariant*: *Resultant*
- Use resultant to compute a modular GCD algorithm for two polynomials over $\mathbb{Z}[x]$
- Allowed us to reduce the problem above to the Euclidean GCD algorithm

Euclidean algorithm only works for Euclidean Domains, and $\mathbb{Z}[x]$ is <u>not</u> an Euclidean domain

gcd in $\mathbb{Z}[x]$ $\longleftrightarrow$ gcd $\underbrace{\mathbb{F}_{p_i}[x]}_{\text{Euclidean domains}}$

EEA

Resultants and Polynomial GCD

- GCD between two polynomials can be characterized by *algebraic invariant*: *Resultant*
- Use resultant to compute a modular GCD algorithm for two polynomials over $\mathbb{Z}[x]$
- Allowed us to reduce the problem above to the Euclidean GCD algorithm

  Euclidean algorithm only works for Euclidean Domains, and $\mathbb{Z}[x]$ is not an Euclidean domain

- Resultants also have nice *theoretical* properties
  1. Identifies the bad primes in modular algorithms
  2. Used as subroutine in factoring algorithms - when double roots appear
  3. Also used to prove upper bound in complexity of ideal membership problem!
  4. Many more applications!

Polynomial Factoring

- Univariate polynomials over *Finite Fields*
  - Cantor-Zassenhaus algorithm
  - Berlekamp-Rabin algorithm

[1] Lovasz won the Abel prize this year - the Nobel prize for mathematics. In their acknowledgments, they mentioned this algorithm as one of his (many) remarkable works!

Polynomial Factoring

- Univariate polynomials over *Finite Fields*
  - Cantor-Zassenhaus algorithm
  - Berlekamp-Rabin algorithm
- Widely used in coding theory!

Berlekamp's decoding algorithm of Reed-Solomon Codes!

[1] Lovasz won the Abel prize this year - the Nobel prize for mathematics. In their acknowledgments, they mentioned this algorithm as one of his (many) remarkable works!

Polynomial Factoring

- Univariate polynomials over *Finite Fields*
  - Cantor-Zassenhaus algorithm
  - Berlekamp-Rabin algorithm
- Widely used in coding theory!

  Berlekamp's decoding algorithm of Reed-Solomon Codes!
- Univariate polynomials over *Integers*
  - Lenstra-Lenstra-Lovasz[1] shortest vector in a lattice algorithm
  - Hensel lifting
  - Bounds on coefficients of factors

[1] Lovasz won the Abel prize this year - the Nobel prize for mathematics. In their acknowledgments, they mentioned this algorithm as one of his (many) remarkable works!

Polynomial Factoring

- Univariate polynomials over *Finite Fields*
  - Cantor-Zassenhaus algorithm
  - Berlekamp-Rabin algorithm
- Widely used in coding theory!

  Berlekamp's decoding algorithm of Reed-Solomon Codes!
- Univariate polynomials over *Integers*
  - Lenstra-Lenstra-Lovasz[1] shortest vector in a lattice algorithm
  - Hensel lifting
  - Bounds on coefficients of factors
- Lattice algorithm can be used to break some cryptosystems, and other applications in number theory!

[1] Lovasz won the Abel prize this year - the Nobel prize for mathematics. In their acknowledgments, they mentioned this algorithm as one of his (many) remarkable works!

Polynomial Factoring

- Univariate polynomials over *Finite Fields*
  - Cantor-Zassenhaus algorithm
  - Berlekamp-Rabin algorithm
- Widely used in coding theory!

  Berlekamp's decoding algorithm of Reed-Solomon Codes!
- Univariate polynomials over *Integers*
  - Lenstra-Lenstra-Lovasz[1] shortest vector in a lattice algorithm
  - Hensel lifting
  - Bounds on coefficients of factors
- Lattice algorithm can be used to break some cryptosystems, and other applications in number theory!
- Factoring *bivariate polynomials*
  - Reduce to univariate factorization
  - Use Hensel lifting to recover multivariate factorization

[1] Lovasz won the Abel prize this year - the Nobel prize for mathematics. In their acknowledgments, they mentioned this algorithm as one of his (many) remarkable works!

Polynomial Factoring

- Univariate polynomials over *Finite Fields*
  - Cantor-Zassenhaus algorithm
  - Berlekamp-Rabin algorithm
- Widely used in coding theory!

  Berlekamp's decoding algorithm of Reed-Solomon Codes!
- Univariate polynomials over *Integers*
  - Lenstra-Lenstra-Lovasz[1] shortest vector in a lattice algorithm
  - Hensel lifting
  - Bounds on coefficients of factors
- Lattice algorithm can be used to break some cryptosystems, and other applications in number theory!
- Factoring *bivariate polynomials*
  - Reduce to univariate factorization
  - Use Hensel lifting to recover multivariate factorization
- Applications in list decoding of Reed-Solomon codes!

[1] Lovasz won the Abel prize this year - the Nobel prize for mathematics. In their acknowledgments, they mentioned this algorithm as one of his (many) remarkable works!

- Administrivia
- Foundations of Symbolic Computation
- Computational Linear Algebra
- Modern Computational Algebra
- Computational Invariant Theory
- Topics I wish I had time to cover

Matrix Multiplication

- Arguably the most used operation in practice
- Strassen's algorithm for faster than naive matrix multiplication
- Like Karatsuba, reduce number of multiplications (as addition is cheaper)

Matrix Multiplication

- Arguably the most used operation in practice
- Strassen's algorithm for faster than naive matrix multiplication
- Like Karatsuba, reduce number of multiplications (as addition is cheaper)
- Also saw that *algorithmically* matrix multiplication is a fundamental problem in linear algebra

Matrix Multiplication

- Arguably the most used operation in practice
- Strassen's algorithm for faster than naive matrix multiplication
- Like Karatsuba, reduce number of multiplications (as addition is cheaper)
- Also saw that *algorithmically* matrix multiplication is a fundamental problem in linear algebra
- Exponent of matrix multiplication ubiquitous in computational linear algebra!

One of the major open problems in computer science!

Matrix Multiplication

- Arguably the most used operation in practice
- Strassen's algorithm for faster than naive matrix multiplication
- Like Karatsuba, reduce number of multiplications (as addition is cheaper)
- Also saw that *algorithmically* matrix multiplication is a fundamental problem in linear algebra
- Exponent of matrix multiplication ubiquitous in computational linear algebra!

  One of the major open problems in computer science!
- Deep connections between matrix multiplication and ranks of tensors

Matrix Inversion & Determinant

- Matrix inversion and determinant often used to solve linear systems

Matrix Inversion & Determinant

- Matrix inversion and determinant often used to solve linear systems
- Both problems have *same complexity* as matrix multiplication!

Matrix Inversion & Determinant

- Matrix inversion and determinant often used to solve linear systems
- Both problems have *same complexity* as matrix multiplication!
- To compute the determinant quickly, had to learn how to compute ALL partial derivatives of a polynomial with same complexity as computing the polynomial itself!

Known as *backpropagation* in Machine Learning.

Matrix Inversion & Determinant

- Matrix inversion and determinant often used to solve linear systems
- Both problems have *same complexity* as matrix multiplication!
- To compute the determinant quickly, had to learn how to compute ALL partial derivatives of a polynomial with same complexity as computing the polynomial itself!

  Known as *backpropagation* in Machine Learning.
- Good thing about the recurrence we found is that *parallel algorithms* for matrix multiplication yield *parallel algorithms* for matrix inversion and determinant!

Black-Box Linear Algebra

- An ubiquitous problem in scientific computing is to solve system of linear equations $A\mathbf{y} = \mathbf{b}$
    1. linear programming
    2. optimization
    3. polynomial multiplication
    4. factoring
    5. polynomial interpolation (DFT)
    6. computing GCD of polynomials (Resultants)
    7. many more

Black-Box Linear Algebra

- An ubiquitous problem in scientific computing is to solve system of linear equations $A\mathbf{y} = \mathbf{b}$
  1. linear programming
  2. optimization
  3. polynomial multiplication
  4. factoring
  5. polynomial interpolation (DFT)
  6. computing GCD of polynomials (Resultants)
  7. many more
- Often times, the input matrix A has very *special structure*, can exploit this structure to obtain *faster algorithms*

Black-Box Linear Algebra

- An ubiquitous problem in scientific computing is to solve system of linear equations $A\mathbf{y} = \mathbf{b}$
    1. linear programming
    2. optimization
    3. polynomial multiplication
    4. factoring
    5. polynomial interpolation (DFT)
    6. computing GCD of polynomials (Resultants)
    7. many more
- Often times, the input matrix A has very *special structure*, can exploit this structure to obtain *faster algorithms*
- Often times, given a vector $\mathbf{c}$, we can evaluate $A\mathbf{c}$ much faster than the naive $O(n^2)$ algorithm. Can use it to get faster algorithms for linear system solving.

Black-Box Linear Algebra

- An ubiquitous problem in scientific computing is to solve system of linear equations $A\mathbf{y} = \mathbf{b}$
  1. linear programming
  2. optimization
  3. polynomial multiplication
  4. factoring
  5. polynomial interpolation (DFT)
  6. computing GCD of polynomials (Resultants)
  7. many more
- Often times, the input matrix A has very *special structure*, can exploit this structure to obtain *faster algorithms*
- Often times, given a vector $\mathbf{c}$, we can evaluate $A\mathbf{c}$ much faster than the naive $O(n^2)$ algorithm. Can use it to get faster algorithms for linear system solving.
- We have already done that many times!

Cost of Evaluation

- Often times, given a vector $\mathbf{b}$, we can evaluate $A\mathbf{b}$ much faster than the naive $O(n^2)$ algorithm. Can use it to get faster algorithms

Cost of Evaluation

- Often times, given a vector $\mathbf{b}$, we can evaluate $A\mathbf{b}$ much faster than the naive $O(n^2)$ algorithm. Can use it to get faster algorithms
- Let $c(A)$ be the cost of multiplying A by any vector $\mathbf{b}$, and $M(n)$ the cost of multiplying two degree n polynomials

Class of matrices	$c(A)$
general	$2n^2 - 2$
Sylvester Matrix (Resultants)	$O(M(n))$
DFT	$O(n \log n)$
Vandermonde matrix	$O(M(n) \log n)$
Berlekamp matrix over $\mathbb{F}_q$	$O(M(n) \log q)$
Sparse matrix with s non-zero entries	$2s$
Toeplitz matrix	$O(M(n))$

Cost of Evaluation

- Often times, given a vector $\mathbf{b}$, we can evaluate $A\mathbf{b}$ much faster than the naive $O(n^2)$ algorithm. Can use it to get faster algorithms
- Let $c(A)$ be the cost of multiplying A by any vector $\mathbf{b}$, and $M(n)$ the cost of multiplying two degree n polynomials

Class of matrices	$c(A)$
general	$2n^2 - 2$
Sylvester Matrix	$O(M(n))$
DFT	$O(n \log n)$
Vandermonde matrix	$O(M(n) \log n)$
Berlekamp matrix over $\mathbb{F}_q$	$O(M(n) \log q)$
Sparse matrix with s non-zero entries	$2s$
Toeplitz matrix	$O(M(n))$

- And these matrices appear quite often in practical applications!

Cost of Evaluation

- Often times, given a vector $\mathbf{b}$, we can evaluate $A\mathbf{b}$ much faster than the naive $O(n^2)$ algorithm. Can use it to get faster algorithms
- Let $c(A)$ be the cost of multiplying A by any vector $\mathbf{b}$, and $M(n)$ the cost of multiplying two degree n polynomials

Class of matrices	$c(A)$
general	$2n^2 - 2$
Sylvester Matrix	$O(M(n))$
DFT	$O(n \log n)$
Vandermonde matrix	$O(M(n) \log n)$
Berlekamp matrix over $\mathbb{F}_q$	$O(M(n) \log q)$
Sparse matrix with s non-zero entries	$2s$
Toeplitz matrix	$O(M(n))$

- And these matrices appear quite often in practical applications!
- In lecture 22 we devised much faster algorithms for inverting matrices with low $c(A)$ in black-box model

- Administrivia
- Foundations of Symbolic Computation
- Computational Linear Algebra
- Modern Computational Algebra
- Computational Invariant Theory
- Topics I wish I had time to cover

Connections Between Algebra & Geometry

- Ideals in $\mathbb{C}[x_1, \ldots, x_n]$ are *finitely generated*

  Hilbert's basis theorem.

Connections Between Algebra & Geometry

- Ideals in $\mathbb{C}[x_1, \ldots, x_n]$ are *finitely generated*

  Hilbert's basis theorem.

- (radical) Ideals in polynomial rings correspond to algebraic sets in finite-dimensional vector spaces

  Hilbert's Nullstellensatz.

Connections Between Algebra & Geometry

- Ideals in $\mathbb{C}[x_1, \ldots, x_n]$ are *finitely generated*

  Hilbert's basis theorem.

- (radical) Ideals in polynomial rings correspond to algebraic sets in finite-dimensional vector spaces

  Hilbert's Nullstellensatz.

- Central problems in modern commutative algebra:
  1. Ideal membership problem — Gröbner bases
  2. Solving System of Polynomial equations — Elimination Theory
  3. Extending partial solutions — Extension Theorem
  4. Implicitization Problem

Applications of Symbolic Commutative Algebra

- Applications in mathematics
  1. Compute dimension of algebraic sets
  2. compute Hilbert Polynomials important numeric invariants
  3. Betti numbers
  4. Resolution of singularities
  5. Many more!

Applications of Symbolic Commutative Algebra

- Applications in mathematics
  1. Compute dimension of algebraic sets
  2. compute Hilbert Polynomials important numeric invariants
  3. Betti numbers
  4. Resolution of singularities
  5. Many more!
- Robotics (robot motion and gemetric descriptions)
- Automatic Geometric Theorem Proving
- Methods to solve integer programming use Gröbner bases

Applications of Symbolic Commutative Algebra

- Applications in mathematics
  1. Compute dimension of algebraic sets
  2. compute Hilbert Polynomials important numeric invariants
  3. Betti numbers
  4. Resolution of singularities
  5. Many more!
- Robotics (robot motion and gemetric descriptions)
- Automatic Geometric Theorem Proving
- Methods to solve integer programming use Gröbner bases
- Bayesian Networks conditional dependencies define algebraic sets!
- Topological data analysis
- many more!

- Administrivia
- Foundations of Symbolic Computation
- Computational Linear Algebra
- Modern Computational Algebra
- Computational Invariant Theory
- Topics I wish I had time to cover

Finite Generation of Rings of Invariants

- We learned that Hilbert himself when he proved the Nullstellensatz and the basis theorem was after proving that

  Ring of invariant polynomials are finitely generated

Finite Generation of Rings of Invariants

- We learned that Hilbert himself when he proved the Nullstellensatz and the basis theorem was after proving that

  Ring of invariant polynomials are finitely generated

- Invariants capture many interesting properties of our algebraic and geometric objects
  1. Whether a matrix is singular or not
  2. bipartite matching
  3. nilpotent matrices
  4. graph isomorphism
  5. word problem over free skew fields
  6. linear matroid intersection
  7. computation of optimal transport distances
  8. contingency tables
  9. Maximum Likelihood Estimation
  10. Symmetries in chemistry molecules
  11. many more

Computational Aspects of Invariant Rings

- Algorithm (via Reynolds operator) to compute invariant polynomials of a certain degree
- Reynolds + Hilbert's argument gave us finite generation of ring of invariants

Computational Aspects of Invariant Rings

- Algorithm (via Reynolds operator) to compute invariant polynomials of a certain degree
- Reynolds + Hilbert's argument gave us finite generation of ring of invariants
- One major open question in computational invariant theory is to *efficiently compute* a generating set of invariants

Computational Aspects of Invariant Rings

- Algorithm (via Reynolds operator) to compute invariant polynomials of a certain degree
- Reynolds + Hilbert's argument gave us finite generation of ring of invariants
- One major open question in computational invariant theory is to *efficiently compute* a generating set of invariants
- Depending on how efficient we can compute the invariants, it can have striking applications in computer science and other fields!

Topics I wish I had time to cover

$R[\bar{x}]$

$\Downarrow$

$\{ p_i(\bar{x}) \geq 0 \}_{i=1}^{t}$

- Solving Differential Equations
- Symbolic Integration
- Semialgebraic Systems of Equations
- Computing Radical of Ideal
- Checking Algebraic Independence
- Computing Primary Decompositions of Ideals } generalization of factoring
- Complexity theory for algebraic computation
- Many more amazing topics in symbolic computation to explore!

Polynomial Identity Testing

finding irreducible components of algebraic sets

Thank you for taking the class!

References I

von zur Gathen, J. and Gerhard, J. 2013.
Modern Computer Algebra
Cambridge University Press